### 

[C++ Solution Template](#_q7jl9546f8ru)

[Common STL usages](#_3n23y8b8gxct)

[Strings](#_rnti3svm8xuw)

[Read line](#_r8skupwml08i)

[Initialization of string](#_80upchboizv5)

[Reverse string](#_gnysr28wvqoc)

[Input C string](#_2xg698lyf2ab)

[Convert char array to string](#_c0idqe5jzenq)

[Swap two strings](#_beerwoa4b6c5)

[Get Substring](#_4qx54ta1gg1y)

[Append](#_efstym6pn7by)

[Compare](#_dxgsqmd49iag)

[find(“string”)](#_hnagjfgtvsff)

[find\_first\_of(“string”)](#_tx9d6x4w4q93)

[find\_last\_of(“string”)](#_uml03crf6ng)

[rfind(“string”)](#_ixl1tn26iqx)

[insert(pos\_to\_begin,string\_to\_insert)](#_8u8fne4698kb)

[Permutation](#_g4iv3dakj7z5)

[std::sort (myints,myints+3);](#_75evyrc7gpek)

[Binary Search](#_so2iy5pdsg5q)

[Upper Bound](#_nwjnbq351myc)

[Lower Bound](#_3xsf5q6hfgjy)

[Sort](#_qcbsvtnkud6l)

[Python](#_hnrfqqltacyo)

[Containers in C++](#_9ostqqcfybg3)

[Sequence containers](#_n1b5efri7owj)

[Heap](#_m0ub1pd2zb27)

[Min-Heap](#_wh1ecak7frds)

[Associative containers](#_52a5nq3fgujz)

[Unordered associative containers](#_aj2xs2r3p3bv)

[Number theory/ Mathematical concepts](#_yeaz0oxsdd87)

[Fast power](#_u9x8bq98ifrr)

[GCD](#_u9x8bq98ifrr)

[Fibonacci using golden ratio](#_u9x8bq98ifrr)

[Almost all concepts](#_u9x8bq98ifrr)

**//done yahan tk**

[Graphs](#_9yk3muo9uqwr)

[Traversal](#_jncc9pkcrfd6)

[Minimum Spanning tree](#_ig0j0gj0wz78)

[Shortest Path](#_q59k0i7i54h3)

[Bipartite Graph](#_yqmghp99b8ri)

[Dynamic Programming](#_mxyqe2r344tr)

[Binary search](#_qyw7kb03sce)

[Miscellaneous](#_tjgt6k95jjku)

[KMP](#_h6ogxoiqcorl)

[Merge Sort](#_ho9abe347jl0)

[Quick sort](#_f4nflzjop5w6)

# 

/\*

Ignore this and previous index page

\*/

# C++ Solution Template

| #include<bits/stdc++.h>  #include<cstdio>  #define pb push\_back  #define mp make\_pair  #define fr first  #define sc second  #define clr(a) memset(a, 0, sizeof(a))1  #define sz(x) x.size()  #define rep(n) for(int i = 0 ; i < n ;i ++)  #define repc(i, n) for(int i = 0 ; i < n ; i ++)  #define FOR(i, x, y) for(int i = x ; i < y ; i ++)  #define DEC(i, x, y) for(int i = x ; i >= y ; i --)  #define all(v) v.begin(), v.end()  #define min3(a, b, c) min(a, min(b,c))  #define max3(a, b, c) max(a, max(b, c))  #define alla(a, n) a, a+n  using namespace std;  // Some typedef's  typedef long long ll;  typedef unsigned long long ull;  typedef pair<ll,ll> ii;  typedef vector<ll> vi;  typedef vector<ii> vii;    // UTILITY FUNCTIONS  // String  char toUpper(const char &c) {return isupper(c)?c:c-'a'+'A';}  char toLower(const char &c) {return islower(c)?c:c+'a'-'A';}  bool isvowel(char c) {c = toLower(c); return (c == 'a' or c == 'e' or c == 'i' or c == 'i' or c == 'o' or c == 'u' || c == 'y');}  void toLowerCase(string &s){rep(s.size()){if(!islower(s[i])){s[i] = tolower(s[i]);}}}  void toUpperCase(string &s){rep(s.size()){s[i] = toupper(s[i]);}}  bool ispal(string s){ll lo, hi;lo = 0;hi = s.length()-1;while(lo <= hi){if(s[lo] != s[hi]){return 0;}lo++;--hi;}return 1;}  template<class T >string toString(T n){string v = "";while(n){v.pb(n % 10 + '0');n /= 10;}reverse(all(v));return v;} | // Maths  template<typename t> t gcd(t a, t b){a = abs(a), b = abs(b);return ((b == 0) ? a : gcd(b, a%b));}  template<typename t> t lcm(t a, t b){return (a \* (b / gcd(a, b)));}  template <typename T> T modpow(T base, T exp, T modulus) {base %= modulus;T result = 1;while (exp > 0) {if (exp & 1) result = (result \* base) % modulus;base = (base \* base) % modulus;exp >>= 1;}return result;}  template<typename T> void cumulative(T \*a, T \*b, T n){rep(n){i ? b[i] = b[i-1] + a[i] : b[i] = a[i];}}  // Some constants  const int inf = 1e9 + 7;  const int eps = 1e-9;  // Debug  void debug(ll x){  cerr << "Debug: " << x << endl;  }    template<typename T>  void debug(T \*a, int n){  cerr << "Debug: ";  rep(n)cerr << a[i] << " ";  cerr << endl;  }  template<typename T>  void debug(vector<T> &a){  cerr << "Debug: ";  rep(sz(a))cerr << a[i] << " ";  cerr << endl;  } |
| --- | --- |

# Common STL usages

## Strings

### Read line

/\* use cin.ignore() when cin is used before getline() int n;cin>>n;cin.ignore();\*/string s;getline(cin, s);

### Reverse string string s;reverse(s.begin(), s.end());

* + 1. Split string into tokens

vector<string> coll;string s = “Hi I am ABC”;stringstream check1(s);string intermediate;

while(getline(check1, intermediate, ‘ ’)){coll.push\_back(intermediate));}

### Input C string char s[12];gets(s);

### Convert char array to stringchar arrstr[] = "this is a string";string target = string(arr);

### Swap two strings str1.swap(str2); // using swap() to swap string content

### Get Substring

string str="We think in generalities, but we live in details.";

string str2 = **str.substr (3,5);** // "think"

size\_t pos = str.find("live"); // position of "live" in str

string str3 = str.substr (pos); // get from "live" to the end

### Append **str6.append(" extension");** // same as str6 += " extension"

str4.append(str6, 0, 6); // at 0th position 6 character appends part of other tring

### **Compare** returns the difference of second string and first string in integer. **str.compare(str2)**

### find(“string”) Searches the string for the first occurrence of the substring specified in arguments. It returns the position of the first occurrence of substring.

### find\_first\_of(“string”) Searches the string for the first character that matches any of the characters specified in its arguments. It returns the position of the first character that matches.

### find\_last\_of(“string”) Searches the string for the last character that matches any of the characters specified in its arguments. It returns the position of the last character that matches.

### rfind(“string”) Searches the string for the last occurrence of the substring specified in arguments. It returns the position of the last occurrence of substring

### insert(pos\_to\_begin,string\_to\_insert) This function inserts the given substring in the string. It takes two arguments, first the position from which you want to insert the substring and second the substring

## Permutation

bool next\_permutation (BidirectionalIterator first, BidirectionalIterator last);  
bool next\_permutation (BidirectionalIterator first, BidirectionalIterator last, Compare comp);

Example int myints[] = {1,2,3};std::sort (myints,myints+3);

do {std::cout << myints[0] << ' ' << myints[1] << ' ' << myints[2] << '\n';  
 } while ( std::next\_permutation(myints,myints+3) );

## Binary Search binary\_search(startaddress, endaddress, valuetofind)

## Upper Bound Returns an iterator pointing to the 1st ele in the range [first,last) which compares ‘>’ val (not=).

## Lower Bound Returns an iterator pointing to the first element in the range [first,last) which does not compare less than val.

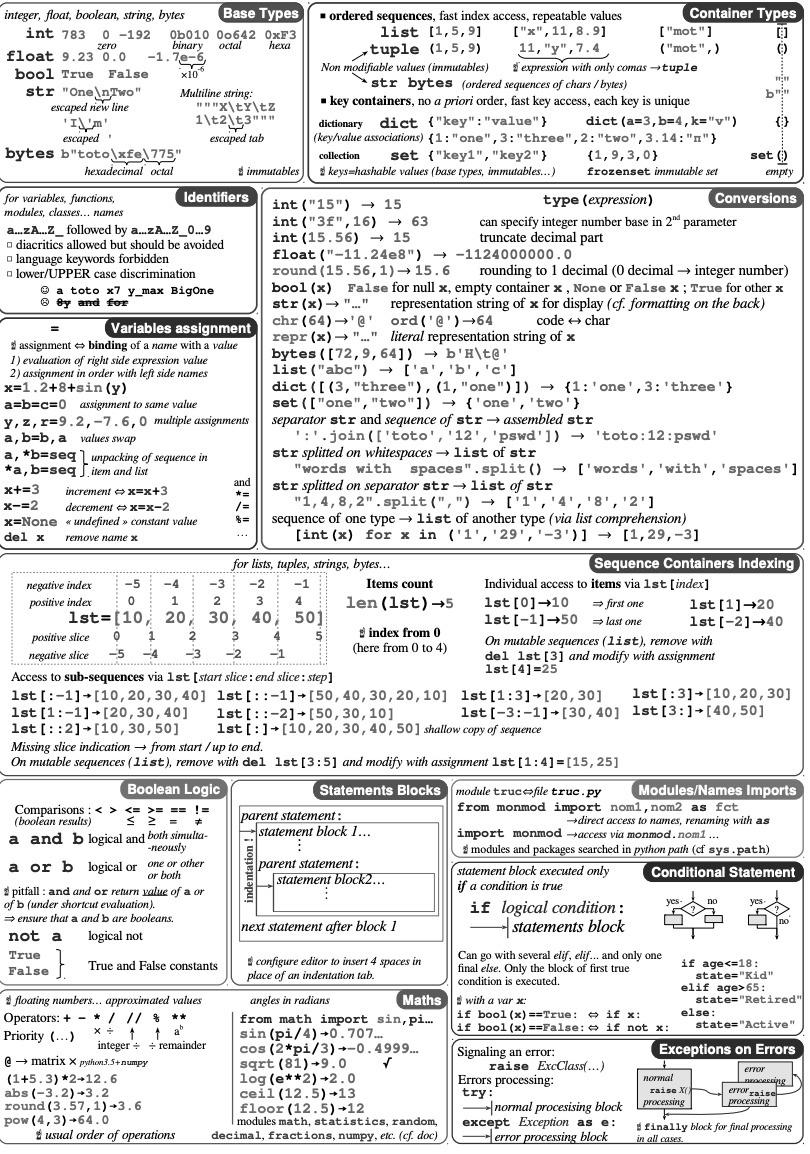
Example: int myints[] = {10,20,30,30,20,10,10,20};  
 std::vector<int> v(myints,myints+8);std::sort (v.begin(), v.end()); std::vector<int>::iterator low,up;  
 low=std::lower\_bound (v.begin(), v.end(), 20); up= std::upper\_bound (v.begin(), v.end(), 20);  
 cout << "lower\_bound at pos" << (low- v.begin()) << '\n'<< "upper\_bound at pos " << (up - v.begin()) << '\n';

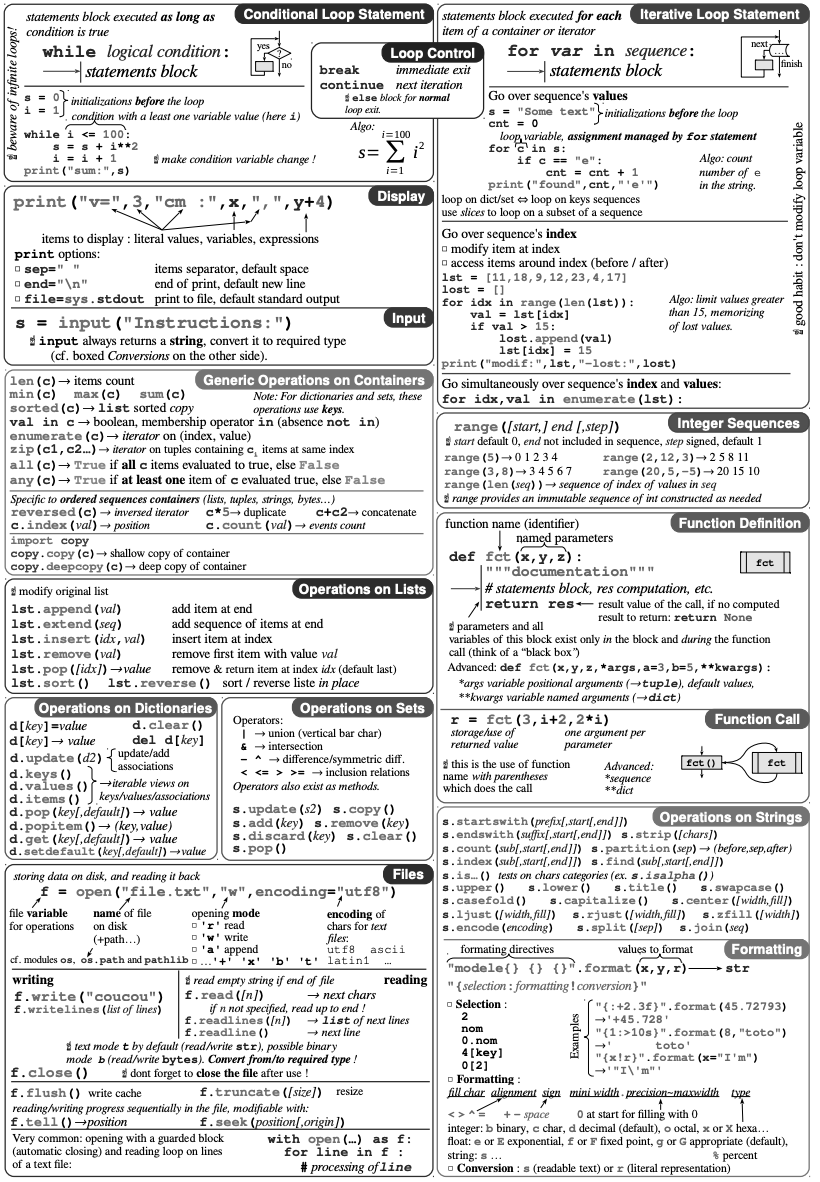
Output: lower\_bound at position 3 upper\_bound at position 6

## Sort

void sort (RandomAccessIterator first, RandomAccessIterator last, Compare comp);

void stable\_sort ( RandomAccessIterator first, RandomAccessIterator last, Compare comp );





# Containers in C++

## Sequence containers

Sequence containers implement data structures which can be accessed sequentially.

* **array**  : Static contiguous array (class template)

ar.fill(0); //fill array ar with value 0

* **vector**

v.insert(v.begin(), 5);// inserts 5 at the beginning

v.erase(v.begin());// removes the first element

* **deque** : Double-ended queue **forward\_list :** Singly-linked list **list** : Doubly-linked list

## Heap

* + 1. **make\_heap**: Rearranges the elements in the range [first,last) in such a way that they form a heap. The element with the highest value is always pointed by first.
    2. **pop\_heap**: Rearranges the elements in the heap range [first,last) in such a way that the part considered a heap is shortened by one: The element with the highest value is moved to (last-1).
    3. **push\_heap**: Given a heap in the range [first,last-1), this function extends the range considered a heap to [first,last) by placing the value in (last-1) into its corresponding location within it.
    4. **sort\_heap**: Sorts the elements in the heap range [first,last) into ascending order. After this array is no longer a heap
    5. **front():** This function displays the first element of heap which is the maximum number.
    6. **is\_heap()** :- This function is used to check whether the container is heap or not. Generally, in most implementations, the reverse sorted container is considered as heap. Returns true if container is heap else returns false.
    7. Min-Heap

Use Priority\_queue

priority\_queue supports a constructor that requires two extra arguments to make it min heap.

priority\_queue <Type, vector<Type>, ComparisonType > min\_heap;

int main () {

priority\_queue <int, vector<int>, greater<int> > pq; // Creates a min heap

pq.push(5); pq.push(1); pq.push(10); pq.push(30); pq.push(20);

while (pq.empty() == false) // One by one extract items from min heap

{ cout << pq.top() << " "; pq.pop(); } return 0; }

MinHeap for user defined class

class Point { int x; int y;

public:

Point(int \_x, int \_y) { x = \_x; y = \_y; }

int getX() const { return x; } nt getY() const { return y; } };

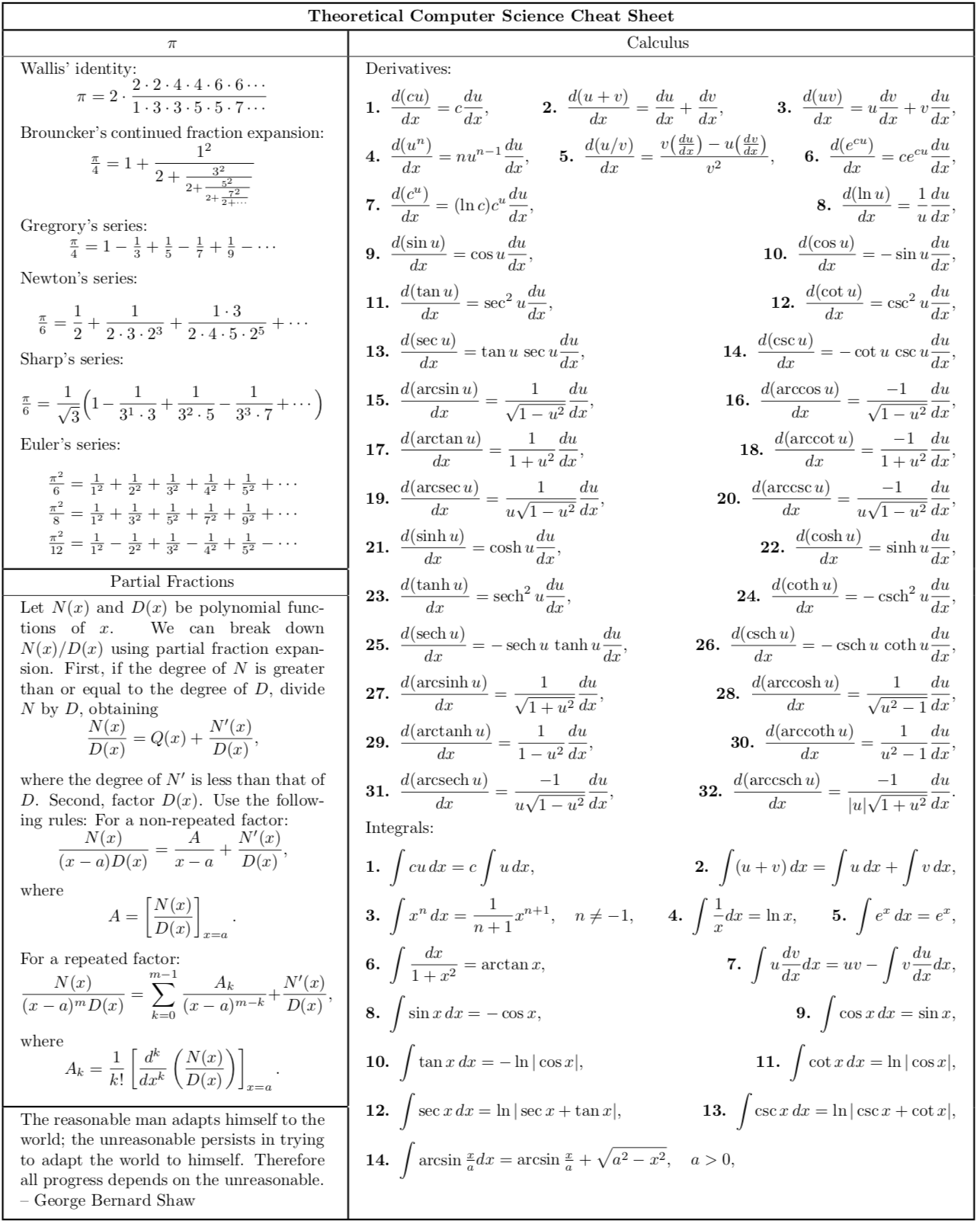
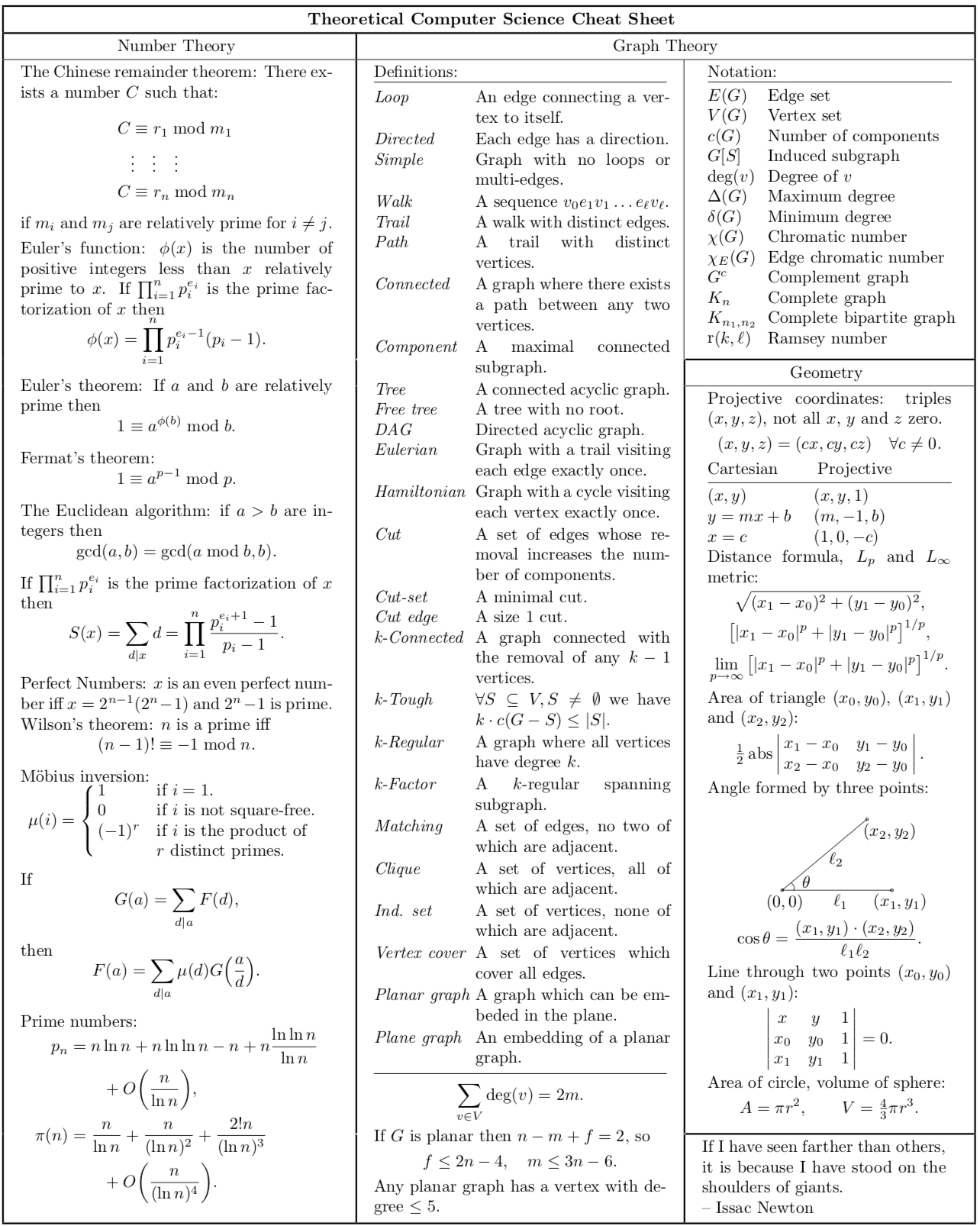
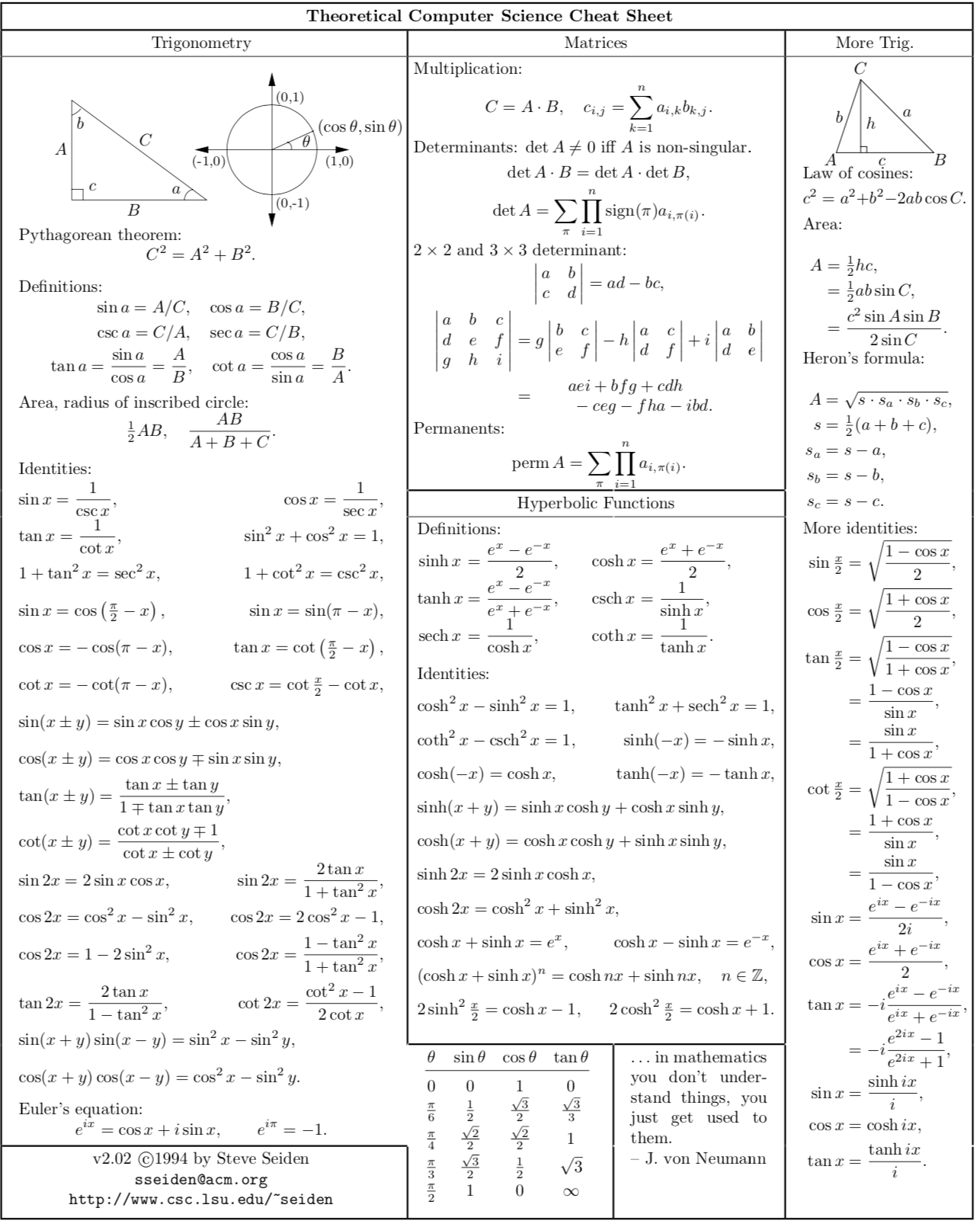
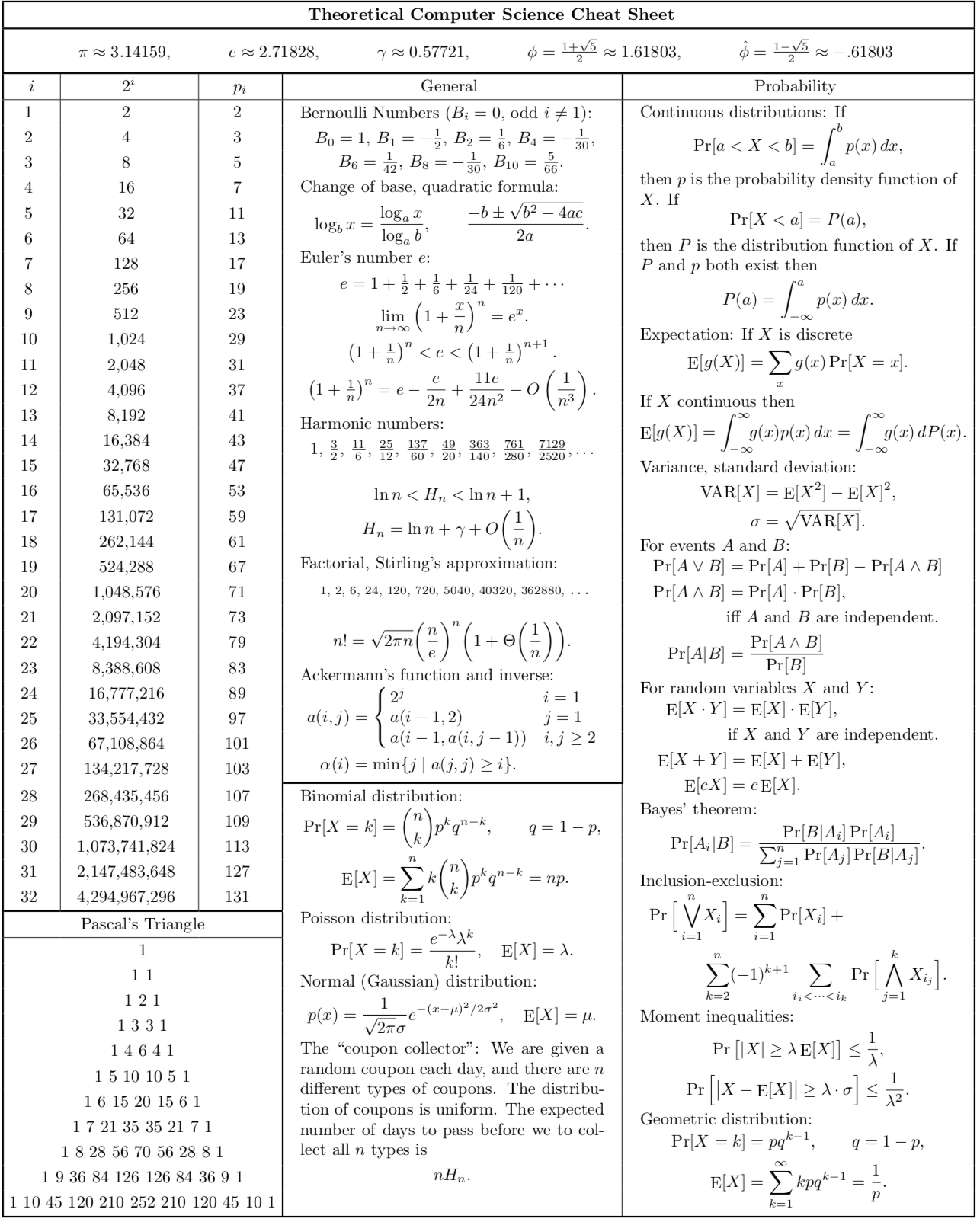
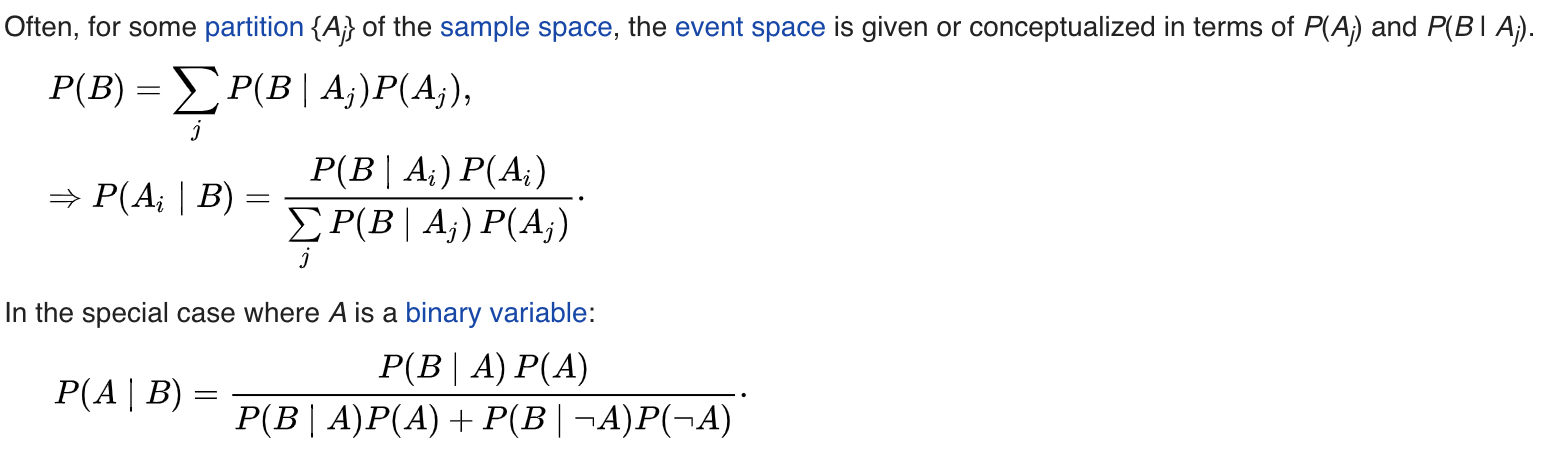
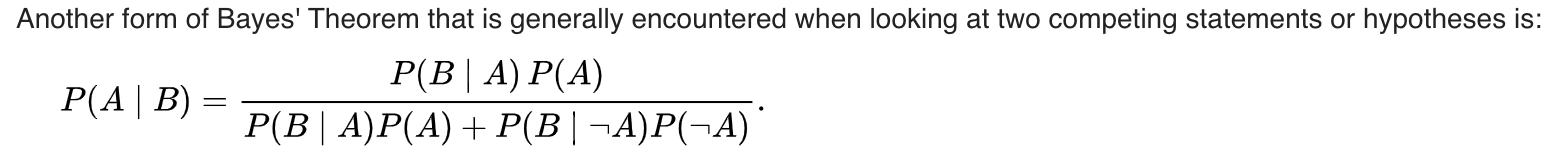
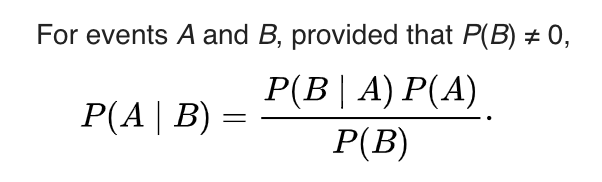
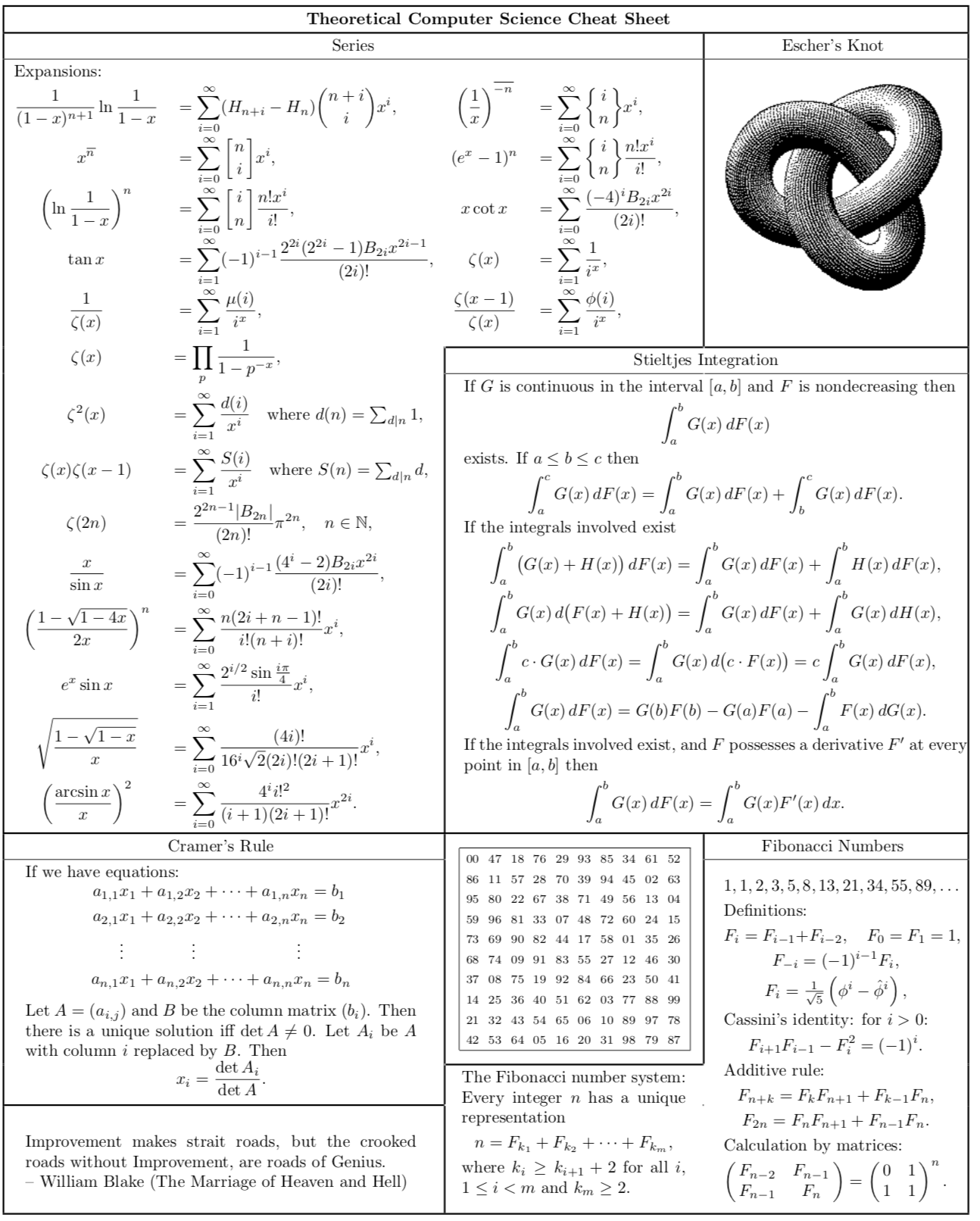
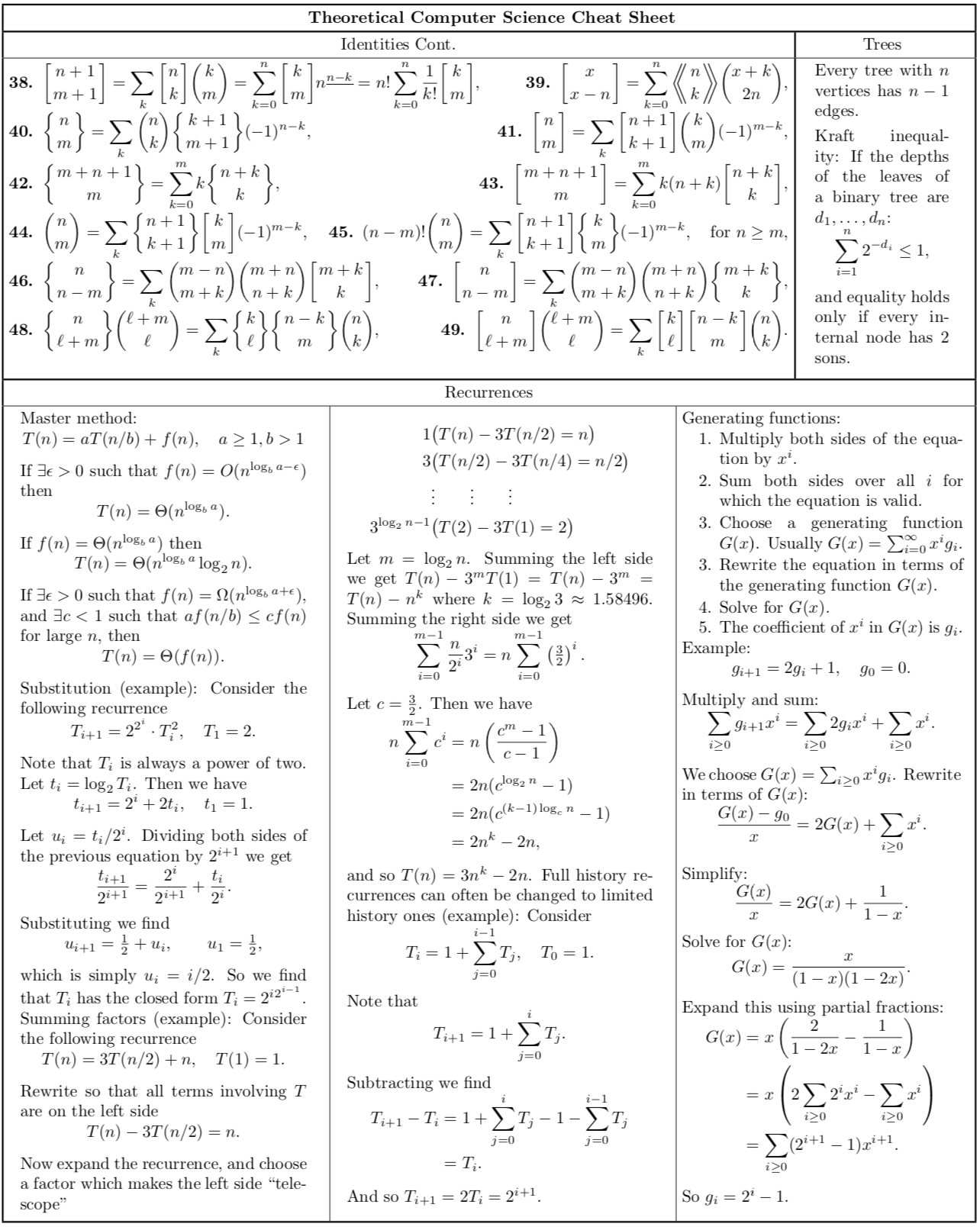
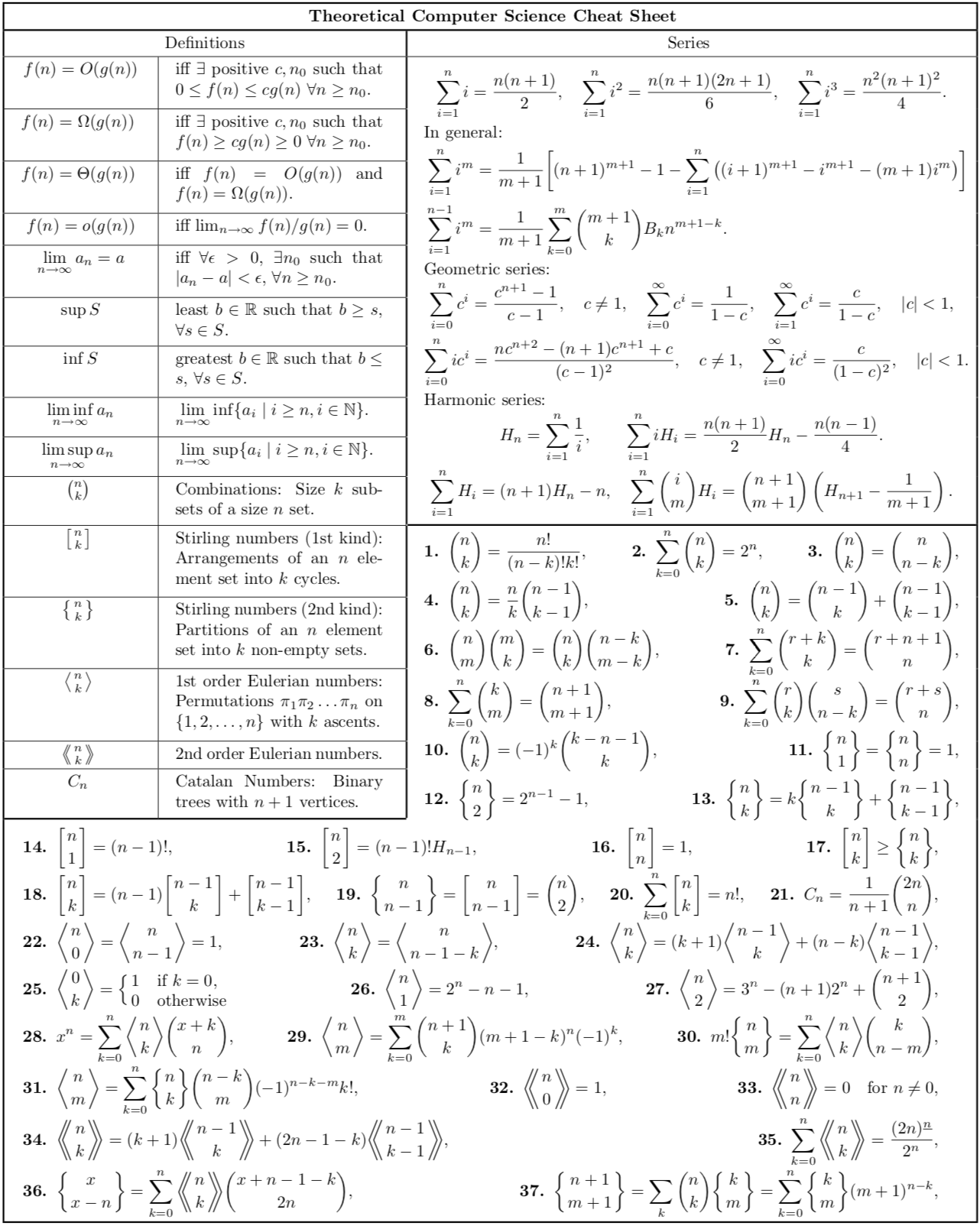
// To compare two points

class myComparator{

public:

int operator() (const Point& p1, const Point& p2){ return p1.getX() > p2.getX();}

};priority\_queue <Point, vector<Point>, myComparator > pq;



GCD:

int **gcd**(int a, int b) { if (a == 0) return b; return gcd(b % a, a); }

**Euler’s Totient function** ?(n) for an input n is count of numbers in {1, 2, 3, …, n} that are relatively prime to n, i.e., the numbers whose GCD (Greatest Common Divisor) with n is 1.

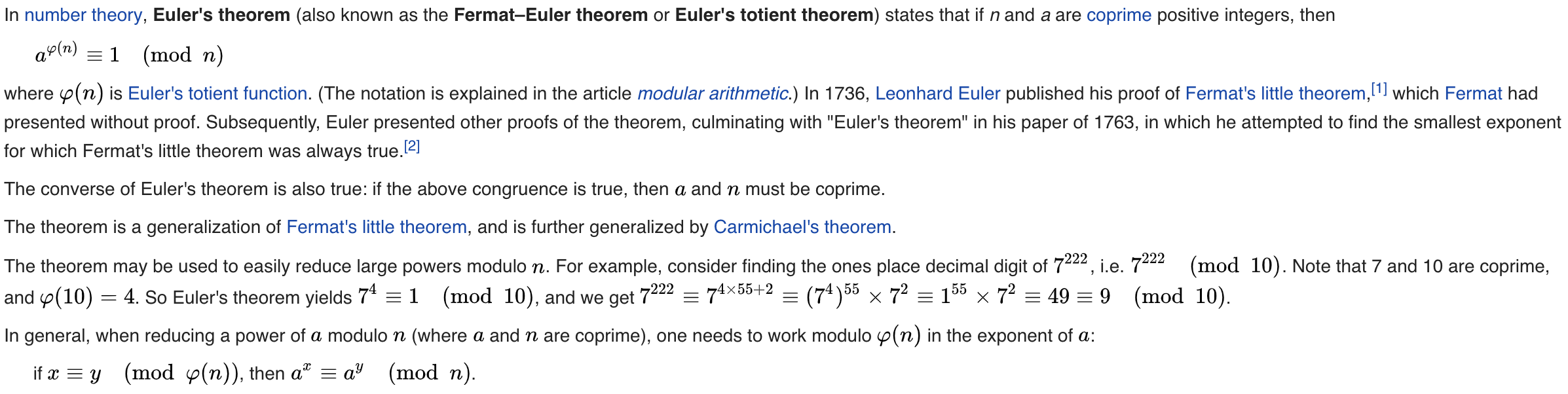
int phi(unsigned int n) {

unsigned int result = 1;

for (int i = 2; i < n; i++) if (gcd(i, n) == 1) result++;

return result;}

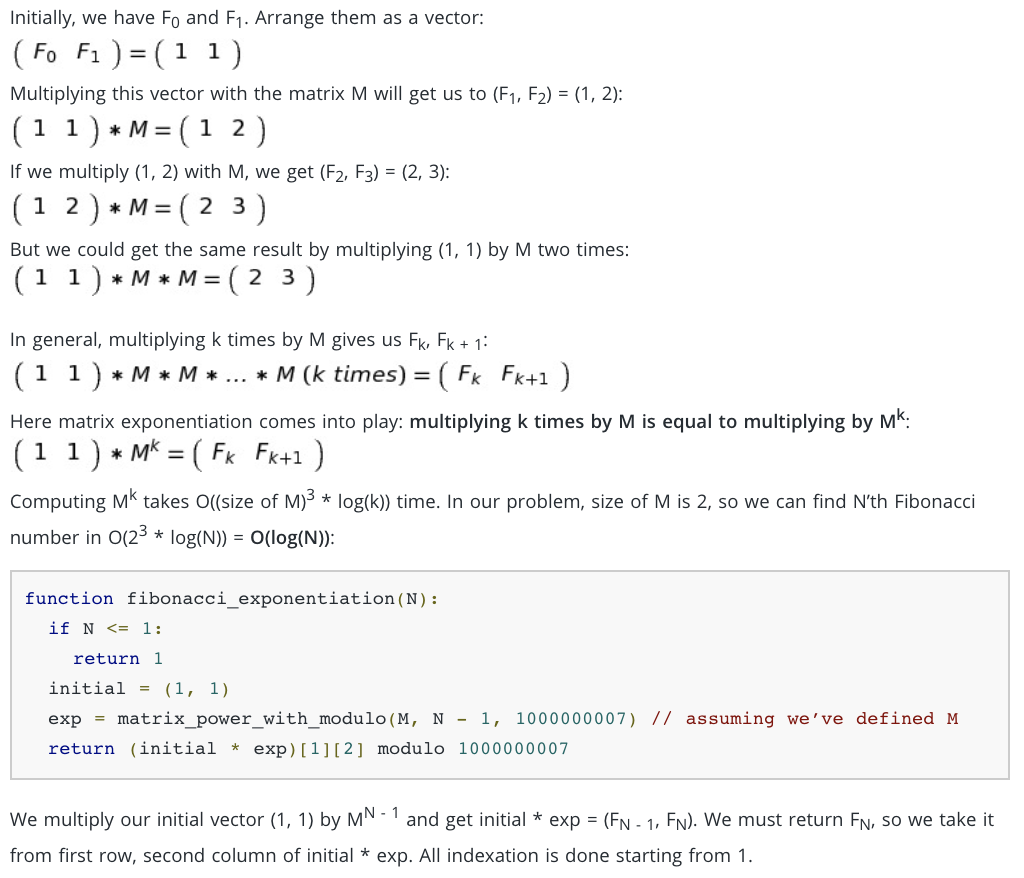
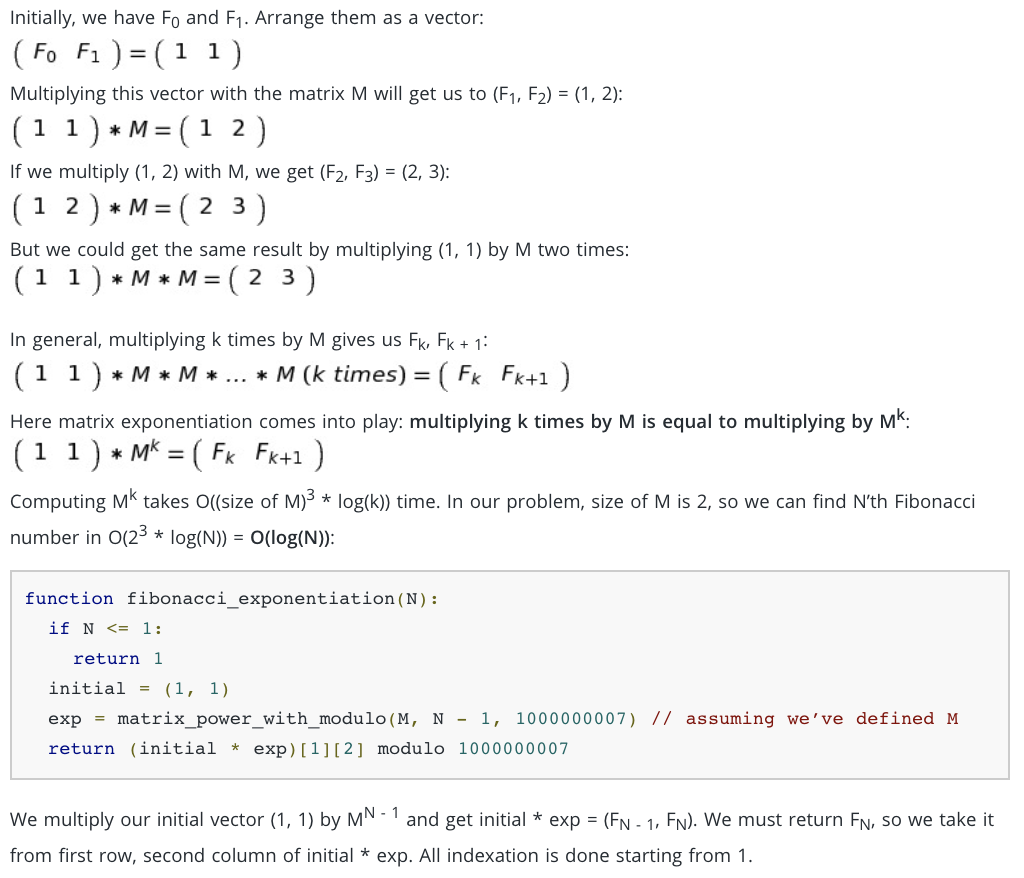
The generalization of Fermat's theorem is known as **Euler's theorem**. In general, Euler's theorem states that, “if p and q are relatively prime, then ”, where φ is Euler's totient function for integers. That is, is the number of non-negative numbers that are less than q and relatively prime to q.



**Multiplicative**

1. P(A ∩ B) = P(A) P(B | A).
2. For two events A and B such that P(B) > 0, P(A | B) ≤ P(A).
3. P(A ∩ B) = P(A) P(B). // independent a b
4. P(A1 ∩ A2 ∩ … ∩ An) = P(A1) P(A2 | A1) P(A3 | A1 ∩ A2) … × P(An |A1 ∩ A2 ∩ … ∩ An-1)
5. P(A1 ∩ A2 ∩ … ∩ An) = P(A1) P(A2) … P(An).

**Matrix exponentiation**



# **Graphs**

vector<int> Adj[V]

## Traversal

| **DFS** | **BFS** |
| --- | --- |
| vector<int> adj[v];  int visited[v];  for(int i=0;i<v;i++){  if(visited[i]==0)  dfs(adj, visited, 0)) }  void dfs(vector<int> adj[], int visited[], int s)  { visited[s]=1;  for(int i=0;i<adj[s].size();i++)  {if(visited[adj[s][i]]==0)  dfs(adj,visited,adj[s][i]))  }  } | void BFS(int s)  {  bool \*visited = new bool[V];  for(int i = 0; i < V; i++)  visited[i] = false;  list<int> queue;  visited[s] = true;  queue.push\_back(s);  list<int>::iterator i;  while(!queue.empty()){  s = queue.front();  queue.pop\_front();  for (i = adj[s].begin(); i != adj[s].end(); ++i){  if (!visited[\*i]) {  visited[\*i] = true;  queue.push\_back(\*i); }}}} |

## Minimum Spanning tree

| **Prim’s Algo-**O((V + E)log(V)) | **Kruskal’s Algo -** Elog(E) + Elog(V) |
| --- | --- |
| typedef pair<int, int> iPair;  void primMST(vector<pair<int,int> > adj[], int V)  {  // Create a priority queue to store vertices  priority\_queue< iPair, vector <iPair> , greater<iPair> > pq;  int src = 0;  vector<int> key(V, INF);  vector<int> parent(V, -1);  vector<bool> inMST(V, false);  pq.push(make\_pair(0, src));  key[src] = 0;  /\* Looping till priority queue becomes empty \*/  while (!pq.empty())  {  // The first vertex in pair is the minimum key  // vertex, extract it from priority queue.  // vertex label is stored in second of pair (it  // has to be done this way to keep the vertices  // sorted key (key must be first item  // in pair)  int u = pq.top().second;  pq.pop();  inMST[u] = true; // Include vertex in MST  for (auto x : adj[u]) // Traverse all adjacent of u  {  int v = x.first; // Get vertex label  int weight = x.second; //weight of current.  // If v is not in MST and weight of (u,v) is smaller  // than current key of v  if (inMST[v] == false && key[v] > weight)  {  key[v] = weight;  pq.push(make\_pair(key[v], v));  parent[v] = u;  }  }  }  // Print edges of MST using parent array  for (int i = 1; i < V; ++i)  printf("%d - %d\n", parent[i], i);  } | vector< pair<int, iPair> > edges;  struct DisjointSets // To represent Disjoint Sets  {  int \*parent, \*rnk;  int n;  DisjointSets(int n) // Constructor.  {  this->n = n;  parent = new int[n+1];  rnk = new int[n+1];  for (int i = 0; i <= n; i++)  {  rnk[i] = 0;  parent[i] = i;  }  }  int find(int u)//Find the parent of a node 'u'  {  /\* Make the parent of the nodes in the path  from u--> parent[u] point to parent[u] \*/  if (u != parent[u])  parent[u] = find(parent[u]);  return parent[u];  }  void merge(int x, int y) // Union by rank  {  x = find(x), y = find(y);  if (rnk[x] > rnk[y]) /\* Make tree with smaller height  a subtree of the other tree \*/  parent[y] = x;  else // If rnk[x] <= rnk[y]  parent[x] = y;  if (rnk[x] == rnk[y])  rnk[y]++;  }  };  int kruskalMST()  {  int mst\_wt = 0;  sort(edges.begin(), edges.end());  DisjointSets ds(V);  vector< pair<int, iPair> >::iterator it;  for (it=edges.begin(); it!=edges.end(); it++) {  int u = it->second.first;  int v = it->second.second;  int set\_u = ds.find(u);  int set\_v = ds.find(v);  // Check if the selected edge is creating cycle  if (set\_u != set\_v)  {  cout << u << " - " << v << endl;  mst\_wt += it->first;  ds.merge(set\_u, set\_v);  }  }    return mst\_wt; } |

## Shortest Path

| **Bellman-Ford O(VE) -ve graphs** | **Dijkstra O(ElogV) +ve only** | **Floyd-Warshall O(V^3)** |
| --- | --- | --- |
| void BellmanFord(struct Graph\* graph, int src){  int V = graph->V;  int E = graph->E;  int dist[V];  for (int i = 0; i < V; i++)  dist[i] = INT\_MAX;  dist[src] = 0;  // Relax all edges |V| - 1 times  for (int i = 1; i <= V-1; i++) {  for (int j = 0; j < E; j++) {  int u = graph->edge[j].src;  int v = graph->edge[j].dest;  int weight = graph->edge[j].weight;  if (dist[u] != INT\_MAX && dist[u] + weight < dist[v])  dist[v] = dist[u] + weight;  }  }  // check for negative-weight cycles.  for (int i = 0; i < E; i++)  {  int u = graph->edge[i].src;  int v = graph->edge[i].dest;  int weight = graph->edge[i].weight;  if (dist[u] != INT\_MAX && dist[u] + weight < dist[v])  printf("Graph contains negative weight cycle");  }  printArr(dist, V);  return;} | void shortestPath(vector<pair<int,int> > adj[], int V, int src)  {  // Create a priority queue to store vertices that are being preprocessed. priority\_queue< iPair, vector <iPair> , greater<iPair> > pq;  vector<int> dist(V, INF); pq.push(make\_pair(0, src));  dist[src] = 0;  while (!pq.empty()) {  int u = pq.top().second;  pq.pop();  for (auto x : adj[u]) {  int v = x.first;  int weight = x.second;  if (dist[v] > dist[u] + weight) {  dist[v] = dist[u] + weight;  pq.push(make\_pair(dist[v], v));  }  }  }    // Print shortest distances stored in dist[]  printf("Vertex Distance from Source\n");  for (int i = 0; i < V; ++i)  printf("%d \t\t %d\n", i, dist[i]);  } | void floydWarshall (int graph[][V])  {  int dist[V][V], i, j, k; //has ouput  for (i = 0; i < V; i++)  for (j = 0; j < V; j++)  dist[i][j] = graph[i][j];  for (k = 0; k < V; k++) //k is interm.  {  for (i = 0; i < V; i++)//i->src  {  for (j = 0; j < V; j++)//j-dest  {  if (dist[i][k] + dist[k][j] < dist[i][j])  dist[i][j] = dist[i][k] + dist[k][j];  }  }  }    // Print the shortest distance matrix  printSolution(dist);  } |

| **Bipartite Graph** | **Topological Sort** |
| --- | --- |
| bool isComponentBipartite(vector<int> adj[], bool visited[], bool color[], int s, bool currColor){  visited[s]=true;color[s]=currColor; bool nextColor = !currColor;  for(int i=0;i<adj[s].size();i++) {  if(!visited[adj[s][i]]) {  if(!isComponentBipartite(adj,visited, color,adj[s][i],nextColor))  return false; }  else {  if(!(color[adj[s][i]]^color[s])) return false;}}  return true;} | void topologicalSortUtil(int v, bool visited[],stack<int> &Stack) {  visited[v] = true;  for(list<int>::iterator i = adj[v].begin(); i != adj[v].end(); ++i)  { if (!visited[\*i])  topologicalSortUtil(\*i, visited, Stack);  }  Stack.push(v); } |

| **Bridge** | **Articulation point** |
| --- | --- |
| void bridgeUtil(int u, bool visited[], int disc[], int low[], int parent[]) {  static int time = 0;  visited[u] = true;  disc[u] = low[u] = ++time;  for (list<int>::iterator i = adj[u].begin(); i != adj[u].end(); ++i) {  int v = \*i;  if (!visited[v]) {  parent[v] = u;  bridgeUtil(v, visited, disc, low, parent);  low[u] = min(low[u], low[v]);//Check if the subtree rooted with v has a connection to one of the ancestors of u  if (low[v] > disc[u]){//If the lowest vertex reachable from subtree under v is below u in DFS tree, then u-v is a bridge  cout << u <<" " << v << endl; }  //Update low value of u for parent function calls.  else if (v != parent[u])  low[u] = min(low[u], disc[v]); } }  void bridge() {  bool \*visited = new bool[V];  int \*disc = new int[V];  int \*low = new int[V];  int \*parent = new int[V];  for (int i = 0; i < V; i++) {  parent[i] = NIL;  visited[i] = false; }  for (int i = 0; i < V; i++)  if (visited[i] == false)  bridgeUtil(i, visited, disc, low, parent);  } | void APUtil(int u, bool visited[], int disc[], int low[], int parent[], bool ap[]) {  static int time = 0;  int children = 0;// Count of children in DFS Tree  visited[u] = true;  disc[u] = low[u] = ++time;  for (list<int>::iterator i = adj[u].begin(); i != adj[u].end(); ++i) {  int v = \*i; // v is current adjacent of u  // If v is not visited yet, then make it a child of u in DFS tree and recur for it  if (!visited[v]) {  children++;  parent[v] = u;  APUtil(v, visited, disc, low, parent, ap);  //if subtree rooted with v has a connection to ancestors of u  low[u] = min(low[u], low[v]);  if (parent[u] == NIL && children > 1)  ap[u] = true;  if (parent[u] != NIL && low[v] >= disc[u])  ap[u] = true; }  else if (v != parent[u])  low[u] = min(low[u], disc[v]); } }  void AP() {  bool \*visited = new bool[V];  int \*disc = new int[V];  int \*low = new int[V];  int \*parent = new int[V];  bool \*ap = new bool[V];  for (int i = 0; i < V; i++) {  parent[i] = NIL;  visited[i] = false;  ap[i] = false; }  for (int i = 0; i < V; i++)  if (visited[i] == false)  APUtil(i, visited, disc, low, parent, ap);  for (int i = 0; i < V; i++)  if (ap[i] == true)  cout << i << " "; } |
| **Eulerian path** | **Hamiltonian circuit** |
| /\* The function returns one of the following values  0 --> If grpah is not Eulerian  1 --> If graph has an Euler path (Semi-Eulerian)  2 --> If graph has an Euler Circuit (Eulerian) \*/  int Graph::isEulerian() {  if (isConnected() == false)// Check if all non-zero degree vertices are connected  return 0;  // Count vertices with odd degree  int odd = 0;  for (int i = 0; i < V; i++)  if (adj[i].size() & 1)  odd++;  // If count is more than 2, then graph is not Eulerian  if (odd > 2)  return 0;  // If odd count is 2, then semi-eulerian.  // If odd count is 0, then eulerian  // Note that odd count can never be 1 for undirected graph  return (odd)? 1 : 2; }  **BINARY SEARCH**  **Is A[x] greater than or equal to the target value?”**      First occurence of yes    Last occurence of No    mid=lo+(hi-lo+1)/2; | bool isSafe(int v, bool graph[V][V], int path[], int pos) {  if (graph [ path[pos-1] ][ v ] == 0)  return false;  for (int i = 0; i < pos; i++)  if (path[i] == v)  return false;  return true; }  bool hamCycleUtil(bool graph[V][V], int path[], int pos) {  if (pos == V) {  if ( graph[ path[pos-1] ][ path[0] ] == 1 )  return true;  else  return false;  }  for (int v = 1; v < V; v++) {  if (isSafe(v, graph, path, pos))  {  path[pos] = v;  if (hamCycleUtil (graph, path, pos+1) == true)  return true;  path[pos] = -1; } }  return false; }  bool hamCycle(bool graph[V][V]) {  int \*path = new int[V];  for (int i = 0; i < V; i++)  path[i] = -1;  path[0] = 0;  if ( hamCycleUtil(graph, path, 1) == false )  {  printf("\nSolution does not exist");  return false;  }  printSolution(path);  return true;  }  void printSolution(int path[])  {  printf ("Solution Exists:"  " Following is one Hamiltonian Cycle \n");  for (int i = 0; i < V; i++)  printf(" %d ", path[i]);  // Let us print the first vertex again to show the complete cycle  printf(" %d ", path[0]);  printf("\n");  } |

| **Kosaraju - strongly connected components** | **Cycle of length n in undirected connected graph** |
| --- | --- |
| bool Graph::isSC() {  bool visited[V];  for (int i = 0; i < V; i++)  visited[i] = false;  DFSUtil(0, visited);  for (int i = 0; i < V; i++)//return false is disconnected  if (visited[i] == false)  return false;  Graph gr = getTranspose();//create reverse graph  for(int i = 0; i < V; i++)  visited[i] = false;  gr.DFSUtil(0, visited);  for (int i = 0; i < V; i++)  if (visited[i] == false)  return false;  return true; } | int countCycles(bool graph[][V], int n)  {  // all vertex are marked un-visited intially.  bool marked[V];  memset(marked, 0, sizeof(marked));  // Searching for cycle by using v-n+1 vertices  int count = 0;  for (int i = 0; i < V - (n - 1); i++) {  DFS(graph, marked, n-1, i, i, count);  // ith vertex is marked as visited and  // will not be visited again.  marked[i] = true;  }  return count/2; } |

# **Dynamic Programming**

| **Catalan Number** (no. of Binary trees) | **Longest Increasing Subsequence O(n^2)** | **Longest Common Subsequence O(n^2)** |
| --- | --- | --- |
| Int Table[1024];  Int catalanNumber(int n){  if(Table[n]!=1){return Table[n];}  Table[n] = 0;  for(int i = 1;i<=n;i++){  Table[n]+=catalanNumber(i-1)\*catalanNumber(n-i); }  Return Table[n];  //In mathematics formula is  Catalan no. = | int lis( int arr[], int n )  { int lis[n];lis[0] = 1;  for (int i = 1; i < n; i++ ) {lis[i] = 1;  for (int j = 0; j < i; j++ ){  if ( arr[i] > arr[j] && lis[i] < lis[j] + 1){  lis[i] = lis[j] + 1;}}}  return \*max\_element(lis, lis+n);} | int lcs(string X, string Y, int m, int n )  {int L[m+1][n+1];int i, j;  for (i=0; i<=m; i++)  { for (j=0; j<=n; j++) {  if (i == 0 || j == 0){L[i][j] = 0;}  else if (X[i-1] == Y[j-1]){L[i][j] = L[i-1][j-1] + 1;}  else{L[i][j] = max(L[i-1][j], L[i][j-1]);} } }  return L[m][n];} |
| **Maximum size sub matrix with all 1’s (SQUARE)** | **Longest Palindrome Subsequence O(n^2)** | **Longest Palindrome Substring O(n^2)** |
| void printMaxSubSquare(bool M[R][C])  {int i,j;int S[R][C];  int max\_of\_s, max\_i, max\_j;  /\* Set first column of S[][]\*/  for(i = 0; i < R; i++){S[i][0] = M[i][0];}  /\* Set first row of S[][]\*/  for(j = 0; j < C; j++){S[0][j] = M[0][j];}  /\* Construct other entries of S[][]\*/  for(i = 1; i < R; i++){  for(j = 1; j < C; j++){  if(M[i][j] == 1){ S[i][j] = min(S[i][j-1], S[i-1][j],S[i-1][j-1]) + 1;}  else{S[i][j] = 0;} } }  /\* Find the maximum entry, and indexes of maximum entry  in S[][] \*/  max\_of\_s = S[0][0]; max\_i=0;max\_j=0;  for(i = 0; i < R; i++){  for(j = 0; j < C; j++){  if(max\_of\_s < S[i][j]){max\_of\_s = S[i][j];  max\_i = i;max\_j = j; } } }  //printf("Maximum size sub-matrix is: \n");  for(i = max\_i; i > max\_i - max\_of\_s;i--){  for(j = max\_j; j > max\_j - max\_of\_s; j--)  {printf("%d ", M[i][j]); }  printf("\n"); } }  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**Minimum jumps to reach End**  int minJumps(int arr[], int n)  {int \*jumps = new int[n]; // jumps[n-1] will hold the result  int i, j; if (n == 0 || arr[0] == 0){ return INT\_MAX;} jumps[0] = 0;  // Find the minimum number of jumps to reach arr[i] from arr[0], and assign this value to jumps[i]  for (i = 1; i < n; i++){ jumps[i] = INT\_MAX;  for (j = 0; j < i; j++) {  if (i <= j + arr[j] && jumps[j] != INT\_MAX){jumps[i] = min(jumps[i], jumps[j] + 1); break;}  } } return jumps[n-1]; } | int lps(string str)  { int n = str.size();int i, j, cl;int L[n][n];  for (i = 0; i < n; i++){L[i][i] = 1;}  for (cl=2; cl<=n; cl++){  for (i=0; i<n-cl+1; i++){ j = i+cl-1;  if (str[i] == str[j] && cl == 2){L[i][j] = 2;}  else if (str[i] == str[j]){L[i][j] = L[i+1][j-1] + 2;}  else{ L[i][j] = max(L[i][j-1], L[i+1][j]);} } }  return L[0][n-1];} [cl is the length of the substring] | int longestPalSubstr(string str )  {int n = strlen( str ); bool table[n][n];  memset(table, 0, sizeof(table));  int maxLength = 1;  for (int i = 0; i < n; ++i){table[i][i] = true;}  int start = 0;  for (int i = 0; i < n-1; ++i){  if (str[i] == str[i+1]){table[i][i+1] = true; start = i;  maxLength = 2;} }  for (int k = 3; k <= n; ++k)  {for (int i = 0; i < n-k+1 ; ++i){int j = i + k - 1;  if (table[i+1][j-1] && str[i] == str[j]){table[i][j] = true;  if (k > maxLength){start = i;maxLength = k;} } } }  printf("Longest palindrome substring is: ");  printSubStr( str, start, start + maxLength - 1 );  return maxLength; // return length of LPS } |
| **Subset Sum** (check if sum is present) O(n^2) | **O(sum) Space -- Subset Sum** O(n^2) |
| bool isSubsetSum(int set[], int n, int sum)  {bool subset[n+1][sum+1];  for (int i = 0; i <= n; i++){subset[i][0] = true;}  for (int i = 1; i <= sum; i++){subset[0][i] = false;}  for (int i = 1; i <= n; i++)  {for (int j = 1; j <= sum; j++){  if(j<set[i-1]){subset[i][j] = subset[i-1][j];}  if (j >= set[i-1]){subset[i][j] = subset[i-1][j] ||  subset[i - 1][j-set[i-1]];} } }  /\* // uncomment this code to print table for (int i = 0; i <= n; i++){  for (int j = 0; j <= sum; j++)  printf ("%4d", subset[i][j]);  printf("\n");  }\*/ return subset[n][sum];} | bool isSubsetSum(int arr[], int n, int sum)  {// The value of subset[i%2][j] will be true if there exists a subset of sum j in arr[0, 1, ...., i-1]  bool subset[2][sum + 1];  for (int i = 0; i <= n; i++) {  for (int j = 0; j <= sum; j++) {  // A subset with sum 0 is always possible  if (j == 0){subset[i % 2][j] = true;}  //If there exists no element no sum is possible  else if (i == 0){subset[i % 2][j] = false;}  else if (arr[i - 1] <= j){  subset[i % 2][j] = subset[(i + 1) % 2]  [j - arr[i - 1]] || subset[(i + 1) % 2][j];  else{subset[i % 2][j] = subset[(i + 1) % 2][j];} } }  return subset[n % 2][sum]; } |
| **Maximum Sum sliding window k** | **Maximum sum sub array** | **Coin Change (Infinite Supply)** |
| void printKMax(int arr[], int n, int k)  { deque<int> Qi(k); int i;  for (i = 0; i < k; ++i){  // For very element, the previous smaller elements are useless so remove them from Qi  while ( (!Qi.empty()) && arr[i] >= arr[Qi.back()]){Qi.pop\_back();}  // Remove from rear Add new element at rear of queue  Qi.push\_back(i);} // Process rest of the elements, i.e.from arr[k] to arr[n-1]  for ( ; i < n; ++i){  // The element at the front of the queue is the largest element of previous window, so print it  cout << arr[Qi.front()] << " ";  while ( (!Qi.empty()) && Qi.front() <= i - k){Qi.pop\_front();}  while ( (!Qi.empty()) && arr[i] >= arr[Qi.back()]){Qi.pop\_back();}  // Add current element at the rear of Qi  Qi.push\_back(i);}  cout << arr[Qi.front()]; } | int maximumSumSubarray(int arr[], int n){  int min\_prefix\_sum = 0;  int res = numeric\_limits<int>::min();  int prefix\_sum[n];  prefix\_sum[0] = arr[0];  for (int i = 1; i < n; i++)  prefix\_sum[i] = prefix\_sum[i - 1] + arr[i];  for (int i = 0; i < n; i++) {  res = max(res, prefix\_sum[i] -  min\_prefix\_sum);  min\_prefix\_sum = min(min\_prefix\_sum,  prefix\_sum[i]);  }  return res; } | int count( int S[], int m, int n )  {  int table[n+1];  memset(table, 0, sizeof(table));  // Base case (If given value is 0)  table[0] = 1;  // Pick all coins one by one and update the table[] values after the index greater than or equal to the value of the picked coin  for(int i=0; i<m; i++)  for(int j=S[i]; j<=n; j++)  table[j] += table[j-S[i]];    return table[n];  } |

# Segment tree

| **Sum of a given range** | **Range Maximum Query** |
| --- | --- |
| **int getMid**(int s, int e) { return s + (e -s)/2; }  st --> Pointer to segment tree  si --> Index of current node in the segment tree.  ss & se --> Starting and ending indexes of the segment represented by current node, i.e., st[si]  qs & qe --> Starting and ending indexes of qry range  **int getSumUtil**(int \*st, int ss, int se, int qs, int qe, int si)  { if (qs <= ss && qe >= se){return st[si];}  if (se < qs || ss > qe){return 0;}  return 0;  int mid = getMid(ss, se);  return getSumUtil(st, ss, mid, qs, qe, 2\*si+1) +  getSumUtil(st, mid+1, se, qs, qe, 2\*si+2); }  i--> index of the element to be updated. This index is in the input array.diff --> Value to be added to all nodes which have i in range  **void updateValueUti**l(int \*st, int ss, int se, int i, int diff, int si){if (i < ss || i > se){return;}  st[si] = st[si] + diff;  if (se != ss){int mid = getMid(ss, se);  updateValueUtil(st, ss, mid, i, diff, 2\*si + 1);  updateValueUtil(st, mid+1, se, i, diff, 2\*si + 2);} }  **void updateValue**(int arr[], int \*st, int n, int i, int new\_val)  {if (i < 0 || i > n-1){printf("Invalid Input");return;}  int diff = new\_val - arr[i];  arr[i] = new\_val;updateValueUtil(st, 0, n-1, i, diff, 0);}  **int getSum**(int \*st, int n, int qs, int qe)  {if (qs < 0 || qe > n-1 || qs > qe){printf("Invalid");return-1;}  return getSumUtil(st, 0, n-1, qs, qe, 0);}  int constructSTUtil(int arr[], int ss, int se, int \*st, int si)  {if (ss == se){st[si] = arr[ss];return arr[ss];}  int mid = getMid(ss, se);  st[si] = constructSTUtil(arr, ss, mid, st, si\*2+1) +  constructSTUtil(arr, mid+1, se, st, si\*2+2);  return st[si];}  **int \*constructST**(int arr[], int n)  {int x = (int)(ceil(log2(n)));int max\_size = 2\*(int)pow(2, x) - 1; int \*st = new int[max\_size];constructSTUtil(arr, 0, n-1, st, 0); return st;} | **int getMid**(int s, int e) {return s + (e - s) / 2;}  st -> Pointer to segment tree  Node -> Index of current node in the segment tree .  ss & se -> Starting and ending indexes of the segmnt  represented by current node, i.e., st[node]  l & r -> Starting and ending indexes of range query  **int MaxUtil**(int\* st, int ss, int se, int l, int r, int node){  if (l <= ss && r >= se){return st[node];}  if (se < l || ss > r){return -1;}  int mid = getMid(ss, se);  return max(MaxUtil(st, ss, mid, l, r,2 \* node + 1),  MaxUtil(st, mid + 1, se, l, r, 2 \* node + 2));}  **void updateValue**(int arr[], int\* st, int ss, int se,int index  int value, int node){if (index < ss || index > se) {  cout << "Invalid Input" << endl; return;}  if (ss == se){arr[index] = value;st[node] = value;}  else {int mid = getMid(ss, se);  if (index >= ss && index <= mid){  updateValue(arr, st,ss,mid, index,value,2\*node+1);}  else{updateValue(arr, st, mid + 1, se,index, value, 2 \* node + 2); st[node] = max(st[2 \* node + 1], st[2 \* node + 2]);} return;}  int getMax(int\* st, int n, int l, int r)  {if (l < 0 || r > n - 1 || l > r) {printf("Invalid Input");return-1; } return MaxUtil(st, 0, n - 1, l, r, 0);}  **int constructSTUtil**(int arr[], int ss, int se,  int\* st, int si){  if (ss == se) {st[si] = arr[ss]; return arr[ss];}  int mid = getMid(ss, se);  st[si] = max(constructSTUtil(arr, ss, mid, st, si\*2 + 1),  **constructSTUtil**(arr, mid + 1, se, st, si \* 2 + 2));  return st[si];}  **int \*constructST**(int arr[], int n)  {int x = (int)(ceil(log2(n)));int max\_size = 2\*(int)pow(2, x) - 1; int \*st = new int[max\_size];constructSTUtil(arr, 0, n-1, st, 0); return st;} |

| **KMP O(n+m)** | **TRIE** |
| --- | --- |
| #define HHH 10003 int ne[HHH]; // next[], if par[i] not matched, jump to i = ne[i] //par → pattern ori → original int kmp(string& par, string& ori) {  ne[0] = -1;  for (int p = ne[0], i = 1; i < par.length(); i++) {  while (p >= 0 && par[p+1] != par[i])  p = ne[p];  if (par[p+1] == par[i])  p++;  ne[i] = p;  }  int match = 0;  for (int p = -1, q = 0; q < ori.length(); q++) {  while (p >= 0 && par[p+1] != ori[q])  p = ne[p];  if (par[p+1] == ori[q])  p++;  if (p + 1 == par.length()) { // match!  p = ne[p];  match++;  }  }  return match; // return number of occurance} | const int ALPHABET\_SIZE = 26;  struct TrieNode  {struct TrieNode \*children[ALPHABET\_SIZE];  //isEndOfWord is true if the node represents end of a word  bool isEndOfWord;};  struct TrieNode \*getNode(void)  {struct TrieNode \*pNode = new TrieNode;  pNode->isEndOfWord = false;  for (int i = 0; i < ALPHABET\_SIZE; i++)  pNode->children[i] = NULL;  return pNode;}  // If not present, inserts key into trie if the key is prefix  of trie node, just marks leaf node  void insert(struct TrieNode \*root, string key)  {struct TrieNode \*pCrawl = root;  for (int i = 0; i < key.length(); i++){  int index = key[i] - 'a';  if (!pCrawl->children[index])  pCrawl->children[index] = getNode();  pCrawl = pCrawl->children[index];}  // mark last node as leaf  pCrawl->isEndOfWord = true;  }// Returns true if key presents in trie, else false  bool search(struct TrieNode \*root, string key)  {struct TrieNode \*pCrawl = root;  for (int i = 0; i < key.length(); i++){  int index = key[i] - 'a';  if (!pCrawl->children[index])  return false;  pCrawl = pCrawl->children[index];}  return (pCrawl != NULL && pCrawl->isEndOfWord);} |
| **Sieve of Eratosthenes** | **Prime Factorization** |
| void SieveOfEratosthenes(int n){ bool prime[n+1];  memset(prime, true, sizeof(prime)); for (int p=2; p\*p<=n; p++)  if (prime[p] == true) { for (int i=p\*p;i<=n; i += p)prime[i]=false; }}  for (int p=2; p<=n; p++)  if (prime[p]) cout << p << " ";} | void primeFactors(int n){  while (n%2 == 0){  printf("%d ", 2);  n = n/2;}  for (int i = 3; i <= sqrt(n); i = i+2){  while (n%i == 0){  printf("%d ", i);  n = n/i;}}  if (n > 2)  printf ("%d ", n)  } |
| Int **binary\_search**(int arr[],int n,int x){  Int low = 0,high=n-1,mid;  while(low<=high){ mid = low+(high-low)/2;  if(arr[mid] == x){return mid;}  Else if(arr[mid]>x){high = mid-1;}  Else{low = mid+1;} return -1;  } | Int **FastPower**(int a,int b,int m){  if(b==0){return 1;}  if(b%2==1){return (a%m\*FastPower(a,b-1,m)%m)%m;}  Else{Int x = FastPower(a,b/2,m)%m;}  Return (x%m\*x%m)%m;} |

**System of linear equations, matrix inverse, determinant:**

