

# ACM/ICPC CheatSheet

## Puzzles

## Contents

<b>1</b>	<b>STL Useful Tips</b>	<b>1</b>
1.1	Common libraries . . . . .	1
1.2	Useful constant . . . . .	2
1.3	Space waster . . . . .	2
1.4	Initialize array with predefined value . . . . .	2
1.5	Modifying sequence operations . . . . .	2
1.6	Merge . . . . .	3
1.7	String . . . . .	3
1.8	Heap . . . . .	3
1.9	Sort . . . . .	4
1.10	Permutations . . . . .	4
1.11	Searching . . . . .	4
1.12	Random algorithm . . . . .	4
<b>2</b>	<b>Number Theory</b>	<b>5</b>
2.1	Max or min . . . . .	5
2.2	Greatest common divisor — GCD . . . . .	5
2.3	Least common multiple — LCM . . . . .	5
2.4	If prime number . . . . .	5
2.5	Leap year . . . . .	5
2.6	$a^b \bmod p$ . . . . .	5
2.7	Factorial mod . . . . .	6
2.8	Generate combinations . . . . .	6
<b>3</b>	<b>Searching Algorithms</b>	<b>7</b>
3.1	Depth first search — DFS . . . . .	7
3.2	Breath first search — BFS . . . . .	7
3.3	Find rank $k$ in array . . . . .	7
<b>4</b>	<b>Dynamic Programming</b>	<b>7</b>
4.1	Knapsack problems . . . . .	7
4.2	Longest common subsequence . . . . .	7
4.3	Maximum submatrix . . . . .	7
<b>5</b>	<b>Trees</b>	<b>7</b>
5.1	Tree representation in array . . . . .	7
5.2	Tree traversal . . . . .	7
<b>6</b>	<b>Graph Theory</b>	<b>7</b>
6.1	Flood fill algorithm . . . . .	7
6.2	SPFA — shortest path . . . . .	7
6.3	Floyd-Warshall algorithm — shortest path of all pairs . . . . .	7
6.4	Prim — minimum spanning tree . . . . .	8
6.5	Eulerian path . . . . .	8
6.6	Topological sort . . . . .	8

## 1 STL Useful Tips

### 1.1 Common libraries

---

```
#include<iostream>
#include<cstdio>
#include<cmath>
#include<string>
#include<queue>
#include<stack>
```

```
#include<vector>
#include<deque> // double ended queue
#include<priority_queue> // priority queue
#include <functional> // for hash
#include<algorithm>
#include<cstdlib> // random
#include<ctime>
#include<sstream>
#include<climits> // all useful constants
```

---

## 1.2 Useful constant

```
INT_MIN
INT_MAX
LONG_MIN
LONG_MAX
LLONG_MIN
LLONG_MAX
(~0u) // infinity (for long and long long)
// use (~0u)>>2 for int.
```

---

## 1.3 Space waster

```
// consider to redefine data types to void data range problem
#define int long long // make everyone long long
#define double long double // make everyone long double

// function definitions

#undef int // main must return int
int main(void)
#define int long long // redefine int

// rest of program
```

---

## 1.4 Initialize array with predefined value

```
// for 1d array, use STL fill_n or fill to initialize array
fill(a, a+size_of_a, value)
fill_n(a, size_of_a, value)
// for 2d array, if want to fill in 0 or -1
memset(a, 0, sizeof(a));
// otherwise, use a loop of fill or fill_n through every a[i]
fill(a[i], a[i]+size_of_ai, value) // from 0 to number of row.
```

---

## 1.5 Modifying sequence operations

```
void copy(first, last, result);
void swap(a,b);
void swap(first1, last1, first2); // swap range
void replace(first, last, old_value, new_value); // replace in range
void replace_if(first, last, pred, new_value); // replace in conditions
// pred can be represented in function
// e.x. bool IsOdd (int i) { return ((i%2)==1); }
void reverse(first, last); // reverse a range of elements
```

```
void reverse_copy(first, last, result); // copy a reverse of range of elements
void random_shuffle(first, last); // using built-in random generator to shuffle array
```

---

## 1.6 Merge

---

```
// merge sorted ranges
void merge(first1, last1, first2, last2, result, comp);
// union of two sorted ranges
void set_union(first1, last1, first2, last2, result, comp);
// intersection of two sorted ranges
void set_intersection(first1, last1, first2, last2, result, comp);
// difference of two sorted ranges
void set_difference((first1, last1, first2, last2, result, comp);
```

---

## 1.7 String

---

```
// Searching
unsigned int find(const string &s2, unsigned int pos1 = 0);
unsigned int rfind(const string &s2, unsigned int pos1 = end);
unsigned int find_first_of(const string &s2, unsigned int pos1 = 0);
unsigned int find_last_of(const string &s2, unsigned int pos1 = end);
unsigned int find_first_not_of(const string &s2, unsigned int pos1 = 0);
unsigned int find_last_not_of(const string &s2, unsigned int pos1 = end);
// Insert, Erase, Replace
string& insert(unsigned int pos1, const string &s2);
string& insert(unsigned int pos1, unsigned int repetitions, char c);
string& erase(unsigned int pos = 0, unsigned int len = npos);
string& replace(unsigned int pos1, unsigned int len1, const string &s2);
string& replace(unsigned int pos1, unsigned int len1, unsigned int repetitions, char c);
// String streams
stringstream s1;
int i = 22;
s1 << "Hello world! " << i;
cout << s1.str() << endl;
```

---

## 1.8 Heap

---

```
template <class RandomAccessIterator>
    void push_heap (RandomAccessIterator first, RandomAccessIterator last);
template <class RandomAccessIterator, class Compare>
    void push_heap (RandomAccessIterator first, RandomAccessIterator last,
                    Compare comp);

template <class RandomAccessIterator>
    void pop_heap (RandomAccessIterator first, RandomAccessIterator last);
template <class RandomAccessIterator, class Compare>
    void pop_heap (RandomAccessIterator first, RandomAccessIterator last,
                    Compare comp);

template <class RandomAccessIterator>
    void make_heap (RandomAccessIterator first, RandomAccessIterator last);
template <class RandomAccessIterator, class Compare>
    void make_heap (RandomAccessIterator first, RandomAccessIterator last,
                    Compare comp );

template <class RandomAccessIterator>
```

```

    void sort_heap (RandomAccessIterator first, RandomAccessIterator last);
template <class RandomAccessIterator, class Compare>
    void sort_heap (RandomAccessIterator first, RandomAccessIterator last,
                    Compare comp);
template <class RandomAccessIterator>
    RandomAccessIterator is_heap_until (RandomAccessIterator first,
                                       RandomAccessIterator last);
template <class RandomAccessIterator, class Compare>
    RandomAccessIterator is_heap_until (RandomAccessIterator first,
                                       RandomAccessIterator last,
                                       Compare comp);

```

---

## 1.9 Sort

```

void sort(iterator first, iterator last);
void sort(iterator first, iterator last, LessThanFunction comp);
void stable_sort(iterator first, iterator last);
void stable_sort(iterator first, iterator last, LessThanFunction comp);
void partial_sort(iterator first, iterator middle, iterator last);
void partial_sort(iterator first, iterator middle, iterator last, LessThanFunction comp);
bool is_sorted(iterator first, iterator last);
bool is_sorted(iterator first, iterator last, LessThanOrEqualFunction comp);
// example for sort, if have array x, start_index, end_index;
sort(x+start_index, x+end_index);

```

---

## 1.10 Permutations

```

bool next_permutation(iterator first, iterator last);
bool next_permutation(iterator first, iterator last, LessThanOrEqualFunction comp);
bool prev_permutation(iterator first, iterator last);
bool prev_permutation(iterator first, iterator last, LessThanOrEqualFunction comp);

```

---

## 1.11 Searching

```

// will return address of iterator, call result as *iterator;
iterator find(iterator first, iterator last, const T &value);
iterator find_if(iterator first, iterator last, const T &value, TestFunction test);
bool binary_search(iterator first, iterator last, const T &value);
bool binary_search(iterator first, iterator last, const T &value, LessThanOrEqualFunction comp);

```

---

## 1.12 Random algorithm

```

srand(time(NULL));
// generate random numbers between [a,b)
rand() % (b - a) + a;
// generate random numbers between [0,b)
rand() % b;
// generate random permutations
random_permutation(anArray, anArray + 10);
random_permutation(aVector, aVector + 10);

```

---

## 2 Number Theory

### 2.1 Max or min

---

```
int max(int a, int b) { return a>b ? a:b; }
int min(int a, int b) { return a<b ? a:b; }
```

---

### 2.2 Greatest common divisor — GCD

---

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

---

### 2.3 Least common multiple — LCM

---

```
int lcm(int a, int b)
{
    return a*b/gcd(a,b);
}
```

---

### 2.4 If prime number

---

```
bool prime(int n)
{
    for (int i=2;i*i<=n;i++)
        if (n%i==0) return false;
    return true;
}
```

---

### 2.5 Leap year

---

```
bool isLeap(int n)
{
    if (n%100==0)
        if (n%400==0) return true;
        else return false;

    if (n%4==0) return true;
    else return false;
}
```

---

### 2.6 $a^b \bmod p$

---

```
long powmod(long base, long exp, long modulus) {
    base %= modulus;
    long result = 1;
    while (exp > 0) {
        if (exp & 1) result = (result * base) % modulus;
        base = (base * base) % modulus;
        exp >>= 1;
    }
}
```

---

```
    return result;
}
```

---

## 2.7 Factorial mod

---

```
//n! mod p
int factmod (int n, int p) {
    long long res = 1;
    while (n > 1) {
        res = (res * powmod (p-1, n/p, p)) % p;
        for (int i=2; i<=n%p; ++i)
            res=(res*i) %p;
        n /= p;
    }
    return int (res % p);
}
```

---

## 2.8 Generate combinations

---

```
// n>=m, choose M numbers from 1 to N.
void combination(int n, int m)
{
    if (n<m) return ;
    int a[50]={0};
    int k=0;

    for (int i=1;i<=m;i++) a[i]=i;
    while (true)
    {
        for (int i=1;i<=m;i++)
            cout << a[i] << " ";
        cout << endl;

        k=m;
        while ((k>0) && (n-a[k]==m-k)) k--;
        if (k==0) break;
        a[k]++;
        for (int i=k+1;i<=m;i++)
            a[i]=a[i-1]+1;
    }
}
```

---

## 3 Searching Algorithms

### 3.1 Depth first search — DFS

### 3.2 Breath first search – BFS

### 3.3 Find rank $k$ in array

## 4 Dynamic Programming

### 4.1 Knapsack problems

### 4.2 Longest common subsequence

### 4.3 Maximum submatrix

## 5 Trees

### 5.1 Tree representation in array

### 5.2 Tree traversal

## 6 Graph Theory

### 6.1 Flood fill algorithm

---

```
//component(i) denotes the
//component that node i is in
void flood_fill(new_component)
do
    num_visited = 0
    for all nodes i
        if component(i) = -2
            num_visited = num_visited + 1
            component(i) = new_component

    for all neighbors j of node i
        if component(j) = nil
            component(j) = -2
until num_visited = 0

void find_components()
    num_components = 0
    for all nodes i
        component(node i) = nil
    for all nodes i
        if component(node i) is nil
            num_components = num_components + 1
            component(i) = -2
            flood_fill(component num_components)
```

---

### 6.2 SPFA — shortest path

### 6.3 Floyd-Warshall algorithm – shortest path of all pairs

---

```
// map[i][j]=infinity at start
void floyd()
{
    for (int k=1; k<=n; k++)
        for (int i=1; i<=n; i++)
```

```
for (int j=1; j<=n; j++)  
    if (i!=j && j!=k && i!=k)  
        if (map[i][k]+map[k][j]<map[i][j])  
            map[i][j]=map[i][k]+map[k][j];  
}
```

---

6.4 Prim — minimum spanning tree

6.5 Eulerian path

6.6 Topological sort