Programming

# Competitive Programming

## Comparing float

Comparing doubles

Always keep a cushion of EPS

double a, b;

if(a==b) not a very good idea

instead, do the following

#define EPS 0.0000001

if(a-EPS<b and a+EPS>b)

## Predict order requirements

1 sec = ( 1-2 )\* 108  operations

X MB = (x/4) \* 106 sized integer arrays

For example, 10 test cases with N = 10000 means O(N2) is required

## range of data types

|  |  |  |  |
| --- | --- | --- | --- |
| Data type | size | in terms of power of 10 ( if unsigned ) | in terms of power of 10 (if signed ) |
| char | 1 | 2.408239965 | 2.10720997 |
| int | 4 | 9.632959861 | 9.331929866 |
| float | 4 |  |  |
| double | 8 |  |  |
|  |  |  |  |
| short int | 2 | 4.816479931 | 4.515449935 |
| long int | 4 | 9.632959861 | 9.331929866 |
| long long int | 8 |  | 18.96488973 |
|  |  |  |  |
| unsigned char | 1 | 2.408239965 |  |
| unsigned short int | 2 | 4.816479931 |  |
| unsigned long int / unsigned int | 4 | 9.632959861 |  |
| unsigned long long int | 8 | 19.26591972 |  |

Char 🡪 short int 🡪int / long int 🡪 long long int

10^2 🡪 10^4 🡪 10^9 🡪 10^19

# STL

Website : http://www.cplusplus.com/reference

1 Data Structures

Vector , stack , queue , priority\_queue ,set , map

2 Algorithms

Find , max, min , sort , reverse , swap

# using bitwise operators

// See if ith bit is set in n

(n & (1 << i)) > 0

String

// Set ith bit of n

n |= (1 << i);

// Reset ith bit of n

n &= ~(1 << i);

// Flip ith bit of n

n ^= (1 << i);

// Get number of ones in n (for gcc)

\_\_builtin\_popcount(n);

// Get m least significant bits of n

n &= (1 << m) - 1;

// Get the least significant set bit

n & (-n)

* To get ith bit of n :

public static boolean GetBit(int n, int index) {

return ((n & (1 << index)) > 0);

}

* To set ith bit of n to b :

public static int SetBit(int n, int index, boolean b) {

if (b) {

return n | (1 << index);

}

else {

int mask = ~(1 << index);

return n & mask;

}

}

In general, we will use an integer to represent a set on a domain of up to 32 values (or 64, using a 64-bit integer), with a 1 bit representing a member that is present and a 0 bit  
one that is absent. Then the following operations are quite straightforward, where ALL\_BITS is a number with 1's for all bits corresponding to the elements of the domain:

Set union

A | B

Set intersection

A & B

Set subtraction

A & ~B

Set negation

ALL\_BITS ^ A

Set bit

A |= 1 << bit

Clear bit

A &= ~(1 << bit)

Test bit

(A & 1 << bit) != 0

Lowest set bit

x & ~(x - 1)

# cumulative sum for many problems

You are given an array **A** of integers of size **N**. You will be given **Q** queries where each query is represented by two integers **L, R**. You have to find the [gcd](http://en.wikipedia.org/wiki/Greatest_common_divisor)(Greatest Common Divisor) of the array after excluding the part from range **L** to **R** inclusive (1 Based indexing). You are guaranteed that after excluding the part of the array  
remaining array is non empty.

e.g. : store cumulative gcd from left and right

// from left front[i] stores

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

{

front[i] = gcd(front[i-1],num[i]);

}

// from right

for(i=n-1;i>=0;i--)

{

back[i] = gcd(back[i+1],num[i]);

}// end of back array

Calculate gcd excluding any rage [l,r]

[printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("%d\n",gcd(front[l],back[r]));

# Millions of strings search for string

1. For storing with anagrams have sorted string as key and all anagrams as values or sum of alphabets mod p (large prime number) as key
2. Store billion phone numbers

cut into pieces, there would be a large number of duplicates in the first two pieces, assuming we are cutting into three pieces.

ISD – STD – unique no   
315-447-9848   
315-447-9851   
  
can be stored into hashtables   
{315}->{447}->...   
  
and use shorts to store the numbers, shorts are two bytes, big enough for 9999

1. For storing billion strings :

If the alphabet size is less or if storage is not much issue or if we want to search for prefixes we can use trie and if we want less storage at the cost of searching then use

ternary search tree or use hashing

<http://stackoverflow.com/questions/7153659/find-an-integer-not-among-four-billion-given-ones>

http://stackoverflow.com/questions/7703049/check-1-billion-cell-phone-numbers-for-duplicates

http://stackoverflow.com/questions/12748246/sorting-1-million-8-digit-numbers-in-1mb-of-ram

<http://stackoverflow.com/questions/7685649/most-efficient-way-to-store-thousand-telephone-numbers>

# Median order statistics

## Binary search

int rank(int[] a , int key) {

int lo = 0;

int hi = a.length - 1;

while (lo <= hi) {

// Key is in a[lo..hi] or not present.

int mid = lo + (hi - lo) / 2;

if(key < a[mid])

hi = mid - 1;

else if (key > a[mid])

lo = mid + 1;

else

return mid;

}

return -1;

}

## Selecting ith smallest in O(n) selecting ith element in A[p … r]

RANDOMIZED-SELECT(*A, p, r, i*)

**if** *p* = *r*

**then return** *A*[*p*]

*q* http://staff.ustc.edu.cn/%7Ecsli/graduate/algorithms/images/arrlt12.gif RANDOMIZED-PARTITION(*A, p, r*)

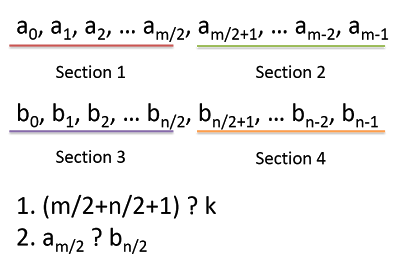
*k* http://staff.ustc.edu.cn/%7Ecsli/graduate/algorithms/images/arrlt12.gif *q* - *p* + 1

if *i* http://staff.ustc.edu.cn/%7Ecsli/graduate/algorithms/images/lteq12.gif *k*

**then return** RANDOMIZED-SELECT(*A, p, q, i*)

**else return** RANDOMIZED-SELECT(*A, q* + 1*, r, i* -k*)*

### Kth element in 2 sorted array



If (m/2+n/2+1) > k && am/2 > bn/2 , drop Section 2

If (m/2+n/2+1) > k && am/2 < bn/2 , drop Section 4

If (m/2+n/2+1) < k && am/2 > bn/2 ,  drop Section 3

If (m/2+n/2+1) < k && am/2 < bn/2 ,  drop Section 1

K nearest points :

<https://github.com/inaoumov/KdTree/blob/master/src/KdTree.java>

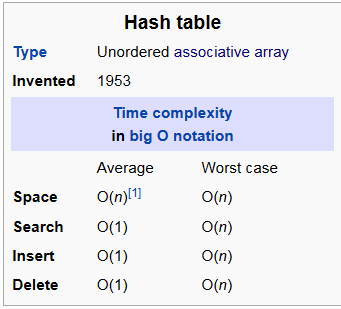
Chess design :

<https://inst.eecs.berkeley.edu/~cs162/sp07/Nachos/chess.shtml>

<https://www3.nd.edu/~cseprog/proj04_212/Chess_Hopkins_Oehmen_Lictenwalter/Final%20Project%20-%20Chess%20Game/cse212final.pdf>

<http://www.pages.drexel.edu/~pv42/thebiz/SDD.pdf>

## Hash table v/s trie :



Hash Table,

Positive :

1. does not require a relative ordering
2. Only require elements are *hashable* and *equality comparable*.

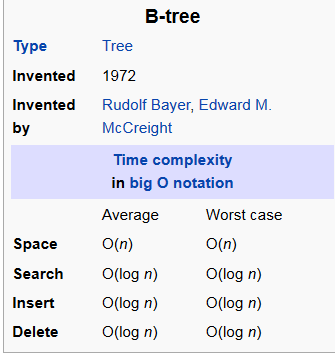
Negative :

1. Worst-case O(n) as compared to log(n)
2. Can’t get sorted order from them

Trie :

1. Can get items in sorted order .
2. Can achieve log(n) using generic data structured like Binary Tree

### B-Tree :



# Greedy

General greedy algorithm

// Pseudocode

public solution greedy(problem) {

solution= empty set;

problem.sort(); // usually place elements in order

for (element: problem) {

if (element feasible and appears optimal)

solution= union(solution, element);

return solution;

}

Some greedy algorithms sort, some use a heap, some may use union-find some don’t need

to sort at all.

# Strings

## character count array

Many problems can be solved using character count array :

int \*getCharCountArray(char \*str)

{

   int \*count = (int \*)calloc(sizeof(int), NO\_OF\_CHARS);

   int i;

   for (i = 0; \*(str+i);  i++)

      count[\*(str+i)]++;

   return count;

}

* Remove all duplicates from the input string.
* Return maximum occurring character in the input string
* Remove characters from the first string which are present in the second string
* Print all the duplicates in the input string.
* Print list items containing all characters of a given word
* Given a string, find its first non-repeating character
* check-whether-two-strings-are-anagram-of-each-other
* Rearrange a string so that all same characters become d distance away

## Important question on strings using brute force

* Length of the longest substring without repeating characters

See LeetCode

* Rabin Karp
* KMP
* Find the smallest window in a string containing all characters of another string (leet code)
* Reverse words in a given string
* <http://www.geeksforgeeks.org/write-a-c-program-to-print-all-permutations-of-a-given-string/>
* Lexicographic rank of a string
* Searching for Patterns | Set 5 (Finite Automata)
* print-all-interleavings-of-given-two-strings
* Print all permutations with repetition of characters
* [Remove “b” and “ac” from a given string](http://www.geeksforgeeks.org/remove-a-and-bc-from-a-given-string/)
* Find the first non-repeating character from a stream of characters (O(1) approach )
* Write your own atoi()

res = res\*10 + str[i] - '0';

* Count words in a given string and Check a given sentence for a given set of simple grammer rules

Use automata with two states

* Anagram Substring Search (Or Search for all permutations)

Modification of rabin karp

* [Given an array of strings, find if the strings can be chained to form a circle](http://www.geeksforgeeks.org/given-array-strings-find-strings-can-chained-form-circle/)

Using directed graph and finding euler circuit

* [Given a sorted dictionary of an alien language, find order of characters](http://www.geeksforgeeks.org/given-sorted-dictionary-find-precedence-characters/)

Using topological sort

* Let 1 represent ‘A’, 2 represents ‘B’, etc. Given a digit sequence, count the number of possible decodings of the given digit sequence.

## Using DP

* Longest Palindromic Substring( better using brute force or suffix tree )
* [Check if a given string is a rotation of a palindrome](http://www.geeksforgeeks.org/check-given-string-rotation-palindrome/)
* Given a string, a partitioning of the string is a palindrome partitioning if every substring of the partition is a palindrome
* Determine the fewest cuts needed for palindrome partitioning of a given string.
* Dynamic Programming | Set 33 (Find if a string is interleaved of two other strings)
* Printing Longest Common Subsequence ( different from Longest Common Substring)
* Longest Even Length Substring such that Sum of First and Second Half is same

## Using Sorting

* given-a-sequence-of-words-print-all-anagrams-together

Instead of sorting the subarray after the ‘first character’, we can reverse the subarray, because the subarray we get after swapping is always sorted in non-increasing order.

## Using recursion

* Print all possible words from phone digits
* print-all-interleavings-of-given-two-strings
* Recursively remove all adjacent duplicates
* Print all permutations with repetition of characters
* Print all possible strings that can be made by placing spaces
* [String matching where one string contains wildcard characters](http://www.geeksforgeeks.org/wildcard-character-matching/)

## Tries , Suffix tree and array

* Longest Common Substring
* Longest palindromic substring
* Given an array of words, print all anagrams together.
* Given a text string and a pattern string, check if pattern exists in text or not.
* Given a text txt[0..n-1] and a pattern pat[0..m-1], write a function search(char pat[], char txt[]) that prints all occurrences of pat[] in txt[].
* Longest repeated substring

Longest repeated substring will end at the internal node which is farthest from the root (i.e. deepest node in the tree), because length of substring is the path label length from root to that internal node.

* Build Linear Time Suffix Array

If we do a DFS traversal, visiting edges in lexicographic order

# BACKTRACKING ALGORITHM

void construct\_candidate(int a[] , int k , int input , int c[] , int \*pnc){

int right = 0;

int down = 0;

\*pnc = 0;

for(int i=0 ; i< k ; i++){

if(a[i] == 0)

right++;

if(a[i] == 1)

down++;

}

if(right < input){

c[\*pnc] = 0;

(\*pnc) = \*pnc + 1;

}

if(down < input){

c[\*pnc] = 1;

(\*pnc) = \*pnc + 1;

}

return;

}

bool is\_solution(int a[] , int k , int input){

if( k == 2\*input)

return true;

else

return false;

}

void process\_solution(int a[] , int k){

printf("{");

for(int i=0 ; i<k ; i++){

printf("%d,", a[i]);

}

printf("}\n");

count++;

}

void backtrack(int a[] , int k , int input){

int c[2];

int nc=0;

if(is\_solution(a,k,input)){

process\_solution(a,k);

}

else{

//k = k + 1;

construct\_candidate(a , k , input , c , &nc);

for(int i=0 ; i < nc ; i++){

a[k] = c[i];

backtrack(a,k+1,input);

}

}

}

# DP problems list

## 1)Learn LIS structure

//Recursive

lld LISRec(lld arr[] , lld n , int\* max\_ref){

lld max\_ending\_here = 1;

for(lld i=1 ; i< n ;i++){

int res = LISRec(arr , i , max\_ref);

if(arr[i-1] < arr[n-1] && max\_ending\_here < res+1){

max\_ending\_here = res + 1;

}

}

if (\*max\_ref < max\_ending\_here)

\*max\_ref = max\_ending\_here;

return maxval;

}

// using DP

lld LISarr[n];

lld parent[n];

for(lld i=1 ; i< n ;i++){

for(lld j=0 ; j< i ; j++){

if(arr[i] > arr[j] && LISarr[i] < LISarr[j] + 1){

LISarr[i] = LISarr[j] + 1;

parent[i] = j;

maxval = std::max(maxval , LISarr[i]);

if(maxval == LISarr[i])

index = i;

}

}

}

return maxval;

Problems :

a)set-3-longest-increasing-subsequence/

b)set-14-maximum-sum-increasing-subsequence/

c)set-15-longest-bitonic-subsequence/

d)set-20-maximum-length-chain-of-pairs/

e)set-21-box-stacking-problem/

f)http://www.geeksforgeeks.org/dynamic-programming-set-14-variations-of-lis/ (IMPORTANT)

## 2)Learn 0-1 Knapsack structure

Problems

a)set-7-coin-change/

b)set-10-0-1-knapsack-problem/

c)set-18-partition-problem/ (Subset Sum)

//Recursive

int count(int val[], int wt[] , int N , int W){

int c = 0;

if(N == 0 || W == 0)

return 0;

if(wt[N-1] >W)

return count(val , wt , N-1 , W) ;

c = std::max(count(val , wt , N-1 , W) , //not taking mth coin

val[N-1] + count(val , wt , N-1 , W - wt[N-1]));//taking mth coin

return c;

}

countDP(int val[], int wt[] , int N , int W){

int table[N+1][W+1]; //one more than no of items

re(i , 0 , N+1)

table[i][0] = 0;

re(i , 0 , W+1)

table[0][i] = 0;

re(i , 1 , N+1){

re(j , 1 , W+1){

if(wt[i-1] > j){

table[i][j] = table[i-1][j];

continue;

}

table[i][j] = std::max(table[i-1][j-wt[i-1]] + val[i-1] , table[i-1][j]);

## 3)Learn Matrix chain multiplication order (very useful ) , especially learn how to fill the matrix

call using count(matrix , 1 , N-1) // N means N-1 matrices

int countmatrixChainREC(int S[],int i , int j){

if(i == j)

return 0;

int c=0 , min =1000000;

re(k , i , j){

c = count(S, i , k) + count(S , k+1 ,j) + S[i-1]\*S[k]\*S[j]; ( (i-1 X k)\*(k X j) )

if(c<min)

min = c;

}

return min;

}

//Usual Structure

matrix-chain-multiplication

int m[n][n];

for (L=2; L<n; L++)

{

for (i=1; i<=n-L+1; i++)

{

j = i+L-1;

m[i][j] = INT\_MAX;

for (k=i; k<=j-1; k++)

{

return m[1][n-1]

longest-palindromic-subsequence/ Palindrome Partitioning)

int n = strlen(str); int n = strlen(str);

int L[n][n]; int C[n][n];

for (L=2; L<=n; L++) for (L=2; L<=n; L++)

{ {

for (i=0; i<n-L+1; i++) // For substring of length L, set different possible starting indexes

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

j = i + L - 1; {

j = i+L-1; // Set ending index

return L[0][n-1]; for (k=i; k<=j-1; k++)

return C[0][n-1];

Problems

a)Set-8-matrix-chain-multiplication/

b)set-11-egg-dropping-puzzle/

c)set-12-longest-palindromic-subsequence/

d)set-17-palindrome-partitioning/

## No Category problems :

Set-4-longest-common-subsequence/

set-5-edit-distance/

set-6-min-cost-path/

set-9-binomial-coefficient/

set-13-cutting-a-rod/

set-16-floyd-warshall-algorithm/

set-18-word-wrap/

set-23-bellman-ford-algorithm/

set-24-optimal-binary-search-tree/

# Inbuilt Search

### Using binary\_search

This version returns only whether an element is present or not , not its position .

Another version returns pointer to element .

### Searching in sorted array (binary\_search() returning only T/F)

#include <iostream> // std::cout

#include <algorithm> // std::binary\_search, std::sort

#include <vector> // std::vector

bool myfunction (int i,int j) { return (i<j); }

int main () {

int myints[] = {1,2,3,4,5,4,3,2,1};

// using default comparison:

std::sort (myints , myints+9);

std::cout << "looking for a 3... ";

if (std::binary\_search (myints,myints+9, 3))

std::cout << "found!\n"; else std::cout << "not found.\n";

// using myfunction as comp:

std::sort (myints,myints+9, myfunction);

std::cout << "looking for a 6... ";

if (std::binary\_search (myints,myints+9, 6, myfunction))

std::cout << "found!\n"; else std::cout << "not found.\n";

return 0;

}

### Searching in sorted array (binary\_search() returning pointer to value)

In C, In C++ there is no binary search which returns pointer to element found

void\* bsearch (const void\* key, const void\* base, size\_t num, size\_t size,

int (\*compar)(const void\*,const void\*));

#include <stdio.h> /\* printf \*/

#include <stdlib.h> /\* qsort, bsearch, NULL \*/

int compareints (const void \* a, const void \* b)

{

return ( \*(int\*)a - \*(int\*)b );

}

int values[] = { 50, 20, 60, 40, 10, 30 };

int main ()

{

int \* pItem;

int key = 40;

qsort (values, 6, sizeof (int), compareints);

pItem = (int\*) bsearch (&key, values, 6, sizeof (int), compareints);

if (pItem!=NULL)

printf ("%d is in the array.\n",\*pItem);

else

printf ("%d is not in the array.\n",key);

return 0;

}

### C++ Binary Search returning element

In practice, lower\_bound tends to be most useful. If [first, last) is a range of iterators and if the elements in the range are sorted in ascending, then

std::lower\_bound(first, last, x);

returns an iterator pointing to an element that's equivalent to x (if such an element exists), or, if there is no such element, an iterator pointing to the position where the element would have been

## Searching in a vector

#include <iostream> // std::cout

#include <algorithm> // std::binary\_search, std::sort

#include <vector> // std::vector

bool myfunction (int i,int j) { return (i<j); }

int main () {

int myints[] = {1,2,3,4,5,4,3,2,1};

std::vector<int> v(myints,myints+9); // 1 2 3 4 5 4 3 2 1

// using default comparison:

std::sort (v.begin(), v.end());

std::cout << "looking for a 3... ";

if (std::binary\_search (v.begin(), v.end(), 3))

std::cout << "found!\n"; else std::cout << "not found.\n";

// using myfunction as comp:

std::sort (v.begin(), v.end(), myfunction);

std::cout << "looking for a 6... ";

if (std::binary\_search (v.begin(), v.end(), 6, myfunction))

std::cout << "found!\n"; else std::cout << "not found.\n";

return 0;

}

## Using find (O(n) Linear search) : finding element position

// find example

there are more find variations

find

find\_end

find\_first\_of

find\_if

find\_if\_not

#include <iostream> // std::cout

#include <algorithm> // std::find

#include <vector> // std::vector

int main () {

// using std::find with array and pointer:

int myints[] = { 10, 20, 30, 40 };

int \* p;

p = std::find (myints, myints+4, 30);

if (p != myints+4)

std::cout << "Element found in myints: " << \*p << '\n';

else

std::cout << "Element not found in myints\n";

// using std::find with vector and iterator:

std::vector<int> myvector (myints,myints+4);

std::vector<int>::iterator it;

it = find (myvector.begin(), myvector.end(), 30);

if (it != myvector.end())

std::cout << "Element found in myvector: " << \*it << '\n';

else

std::cout << "Element not found in myints\n";

return 0;

}

## For searching for a subsequence

\*

// search algorithm example

#include <iostream> // std::cout

#include <algorithm> // std::search

#include <vector> // std::vector

bool mypredicate (int i, int j) {

return (i==j);

}

int main () {

std::vector<int> haystack;

// set some values: haystack: 10 20 30 40 50 60 70 80 90

for (int i=1; i<10; i++) haystack.push\_back(i\*10);

// using default comparison:

int needle1[] = {40,50,60,70};

std::vector<int>::iterator it;

it = std::search (haystack.begin(), haystack.end(), needle1, needle1+4);

if (it!=haystack.end())

std::cout << "needle1 found at position " << (it-haystack.begin()) << '\n';

else

std::cout << "needle1 not found\n";

// using predicate comparison:

int needle2[] = {20,30,50};

it = std::search (haystack.begin(), haystack.end(), needle2, needle2+3, mypredicate);

if (it!=haystack.end())

std::cout << "needle2 found at position " << (it-haystack.begin()) << '\n';

else

std::cout << "needle2 not found\n";

return 0;

}

# Inbuilt Sort

## Sorting Array and Vector

// sort algorithm example

#include <iostream> // std::cout

#include <algorithm> // std::sort

#include <vector> // std::vector

bool myfunction (int i,int j) { return (i<j); }

struct myclass {

bool operator() (int i,int j) { return (i<j);}

} myobject;

int main () {

int myints[] = {32,71,12,45,26,80,53,33};

std::vector<int> myvector (myints, myints+8); // 32 71 12 45 26 80 53 33

//or you can use myints directly

// using default comparison (operator <):

std::sort (myvector.begin(), myvector.begin()+4); //(12 32 45 71)26 80 53 33

// using function as comp

std::sort (myvector.begin()+4, myvector.end(), myfunction); // 12 32 45 71(26 33 53 80)

// using object as comp

std::sort (myvector.begin(), myvector.end(), myobject); //(12 26 32 33 45 53 71 80)

// print out content:

std::cout << "myvector contains:";

for (std::vector<int>::iterator it=myvector.begin(); it!=myvector.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

## Sorting List (non random access )

Use inbuilt sort() function

mylist.sort(my\_comapre\_function);

## Sort\_heap // for sorting heap

// range heap example

#include <iostream> // std::cout

#include <algorithm> // std::make\_heap, std::pop\_heap, std::push\_heap, std::sort\_heap

#include <vector> // std::vector

int main () {

int myints[] = {10,20,30,5,15};

std::vector<int> v(myints,myints+5);

std::make\_heap (v.begin(),v.end());

std::cout << "initial max heap : " << v.front() << '\n';

std::pop\_heap (v.begin(),v.end()); v.pop\_back();

std::cout << "max heap after pop : " << v.front() << '\n';

v.push\_back(99); std::push\_heap (v.begin(),v.end());

std::cout << "max heap after push: " << v.front() << '\n';

std::sort\_heap (v.begin(),v.end());

std::cout << "final sorted range :";

for (unsigned i=0; i<v.size(); i++)

std::cout << ' ' << v[i];

std::cout << '\n';

return 0;

}

## For partitioning vector

template <class BidirectionalIterator, class UnaryPredicate>

BidirectionalIterator partition (BidirectionalIterator first,

BidirectionalIterator last, UnaryPredicate pred);

Partition range in two Rearranges the elements from the range [first,last), in such a way that all the elements for which

pred returns true precede all those for which it returns false. The iterator returned points to the first element of the

second group.

#include <iostream> // std::cout

#include <algorithm> // std::partition

#include <vector> // std::vector

bool IsOdd (int i) { return (i%2)==1; }

int main () {

std::vector<int> myvector;

// set some values:

for (int i=1; i<10; ++i) myvector.push\_back(i); // 1 2 3 4 5 6 7 8 9

std::vector<int>::iterator bound;

bound = std::partition (myvector.begin(), myvector.end(), IsOdd);

// print out content:

std::cout << "odd elements:";

for (std::vector<int>::iterator it=myvector.begin(); it!=bound; ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

std::cout << "even elements:";

for (std::vector<int>::iterator it=bound; it!=myvector.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

## Stable\_partition : To maintain relative ordering use stable\_partition

// stable\_partition example

#include <iostream> // std::cout

#include <algorithm> // std::stable\_partition

#include <vector> // std::vector

bool IsOdd (int i) { return (i%2)==1; }

int main () {

std::vector<int> myvector;

// set some values:

for (int i=1; i<10; ++i) myvector.push\_back(i); // 1 2 3 4 5 6 7 8 9

std::vector<int>::iterator bound;

bound = std::stable\_partition (myvector.begin(), myvector.end(), IsOdd);

// print out content:

std::cout << "odd elements:";

for (std::vector<int>::iterator it=myvector.begin(); it!=bound; ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

std::cout << "even elements:";

for (std::vector<int>::iterator it=bound; it!=myvector.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

## Stable\_sort // maintains relative ordering

// using stable sort

bool compare\_as\_ints (double i,double j)

{

return (int(i)<int(j));

}

int main () {

double mydoubles[] = {3.14, 1.41, 2.72, 4.67, 1.73, 1.32, 1.62, 2.58};

std::vector<double> myvector;

myvector.assign(mydoubles,mydoubles+8);

std::cout << "using default comparison:";

std::stable\_sort (myvector.begin(), myvector.end());

for (std::vector<double>::iterator it=myvector.begin(); it!=myvector.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

myvector.assign(mydoubles,mydoubles+8);

std::cout << "using 'compare\_as\_ints' :";

std::stable\_sort (myvector.begin(), myvector.end(), compare\_as\_ints);

for (std::vector<double>::iterator it=myvector.begin(); it!=myvector.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

O/P

using default comparison: 1.32 1.41 1.62 1.73 2.58 2.72 3.14 4.67

using 'compare\_as\_ints' : 1.41 1.73 1.32 1.62 2.72 2.58 3.14 4.67

## Quick Sort

// using qsort in c++

#include <iostream>

#include <cstdlib>

#include<algorithm>

using namespace std;

#define SIZE 10

int compare (const void \*elem1, const void \*elem2){

return \*(int\*)elem1 - \*(int\*)elem2;

}

bool mycomp(int i , int j){

return i>j;

}

int main(){

int i;

int array[SIZE] = {9,1,3,7,0,5,2,6,8,4};

//using sort

sort(array , array+SIZE , mycomp);

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

std::cout << array[i] << " ";

std::cout << std::endl;

//using qsort

qsort(array, SIZE, sizeof(int), compare);

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

std::cout << array[i] << " ";

std::cout << std::endl;

return 0;

}

## Using Comparable Interface

public interface Comparable< E > {

public int compareTo( E other);

}

Use

return this.x - other.x;

to return values

• A class can implement the Comparable interface to define a natural ordering function for its objects.

• A call to your compareTo method should return:

a value < 0 if the arg comes "before" i.e. less than this one,

a value > 0 if the arg comes "after" i.e. greater than this one,

0 if the arg is considered "equal" to this.

# Misc notes

1)For reference type parameter arguement should be lvalue ,

vector<int> getVector(){

vector<int> v;

return v;

}

vector<int> fillVector(vector<int> v){// no error this will work

cout<<"first";

return v;

}

vector<int> fillVector(vector<int>& v){ // error arguement passed not an l value

cout<<"second";

return v;

}

int main()

{

vector<int> v = fillVector(getVector());

}

O/P

first

if both are prsent then no error as it will call (vector<int> v ) version

2)

NOTE :

memcpy takes size in bytes so to copy 10 integers use :

10\*sizeof(int)

3)

Let's see what O(V+E) time means. Assume for the moment that E≥V, which is the case for most graphs, especially those for which

we run breadth-first search. Then V+∣E∣≤∣E∣+∣E∣=2⋅∣E∣. Because we ignore constant factors in asymptotic notation, we see that

when E≥V, O(V+E) really means O(E). If, however, we have E<∣V∣, then V+∣E∣≤∣V∣+∣V∣=2⋅∣V∣, and so O(V+E) really means O(V).

We can put both cases together by saying that O(V+E) really means O(max(V,E)). In general, if we have parameters x and y, then O(x+y)

really means O(max(x,y)).

4)

initialize static variable before use

class A{

public :

static int var;

}

int A::var = 0;

5) void \* pointer arithmetic can't be performed because void is an incomplete type but still addition and subtraction on void \*

behaves same as char \*

void \* ptr = (void\*)malloc(sizeof(float)\*5);

printf("%d\n",ptr); // print 100

ptr = ptr - 5;

printf("%d\n",ptr); // print 95

6)

std::list<pair<string,int> > not std::list<pair<string,int>>

note the space between the last 2 angular brackets

7)

Deletion in symbol tables generally involves one of two strategies: lazy deletion, where we associate keys in the table with null,

then perhaps remove all such keys at some later time, and eager deletion, where we remove the key from the table immediately.

As just discussed, the code put(key, null) is an easy (lazy) implementation of delete(key). When we give an (eager)

implementation of delete(), it is intended to replace this default.

8)

While this method does the job, it has a flaw that might cause performance problems in some practical situations. The problem is

that the choice of using the successor is arbitrary and not symmetric. Why not use the predecessor?

Since we keep on deleting only from right side in case of successor therefore tree becomes skewed towards left

one solution could be to alternate between left and right depending upon whether tree has even or odd no of nodes

9)

Suppose we wish to repeatedly search a linked list of length N elements, each of which contains a very long string key.

How might we take advantage of the hash value when searching the list for an element with a given key?

Solution: precompute the hash value of each string in the list. When searching for a key t, compare its hash value to the hash

value of a string s. Only compare the string s and t if their hash values are equal.

10)

CLRS page 365

DP has 2 approaches :

a) Top down : in this more no of function calls but need not go to all subproblems i.e. solve all cmaller subproblems

b)bottom up : in this it solves subproblems in increasing order of size , therefore , less function calls but has to solve all

subproblems which may not be required sometimes to get required solution

11)

memset can be used to only set character bytes because it sets characters so can't be used to set integer array

12)For data structures with random access iterators like array , vectors you can use sort(arr) or qsort(v.begin() , v.end())

but for other data structures with no random access like list use class inbuilt function i.e. list.sort()

13)std::binary\_search (myints,myints+9, 3)

just returns whether an element exists or not i.e. bool not its position for finding position use find()

14) instead of keeping start and end just move array pointer and note that adjust the size accordingly

e.g.

if (b[m/2] >= a[n/2]){

if ((n/2 + 1 + m/2) >= k)

return GetMedian(a, n, b, m/2, k); // to reject last half of b

else

return GetMedian(a + n/2 + 1, n - (n/2 + 1), b, m, k - (n/2 + 1)); // to reject first half of a

}

15)

dont directly pass vector function as arguement it will give void error

16)

always clear vector after performing operations since after any operation on vector like insert , clear some iterators become invalid

17) Sometimes const can give error

struct CompareNode{

bool operator() (const ListNode\* &t1 , const ListNode\* &t2){

return (t1->val > t2->val);

}

};

18)for list iterators you can't do

it = it + 1;

it = it - 1;

however you can do it++ and it--

for vector iterators you can do all

19)Always allocate memory using new

not like TreeNode t(10)

but TreeNode t = new TreeNode(10);

Always allocate memory to string first before writing

e.g. string temp;

temp[0] = 'a'

temp[1] = 'b'

it is wrong do

string temp(2,0)

temp[0] = 'a'

temp[1] = 'b'

# Linked List

## Code for merging two linked lists :

ListNode p1 = head ;

ListNode p2 = second;

While(p2 != NULL){

LN temp1 = p1.next;

LN temp2 = p2.next;

P1.next = p2;

P2.next = temp1;

P1= temp1;

P2 = temp2;

}

## Code for reversing linked list

ListNode pre = head;

ListNode curr = head.next;

While(curr != NULL ){

LN temp = curr.next;

Curr.next = pre;

Pre = curr;

Curr = temp;

}

Head.next = NULL ;

| Name | Best | Average | Worst | Memory | Stable | Method |
| --- | --- | --- | --- | --- | --- | --- |
| [Quicksort](http://en.wikipedia.org/wiki/Quicksort) | n \log n | n \log n | n^2 | \log non average, worst case is n; Sedgewick variation is \log nworst case | typical in-place sort is not stable; stable versions exist | Partitioning |
| [Merge sort](http://en.wikipedia.org/wiki/Merge_sort) | n \log n | n \log n | n \log n | nworst case | Yes | Merging |
| [In-place merge sort](http://en.wikipedia.org/wiki/In-place_merge_sort) | — | — | n \log^2 n | 1 | Yes | Merging |
| [Heapsort](http://en.wikipedia.org/wiki/Heapsort) | n \log n | n \log n | n \log n | 1 | No | Selection |
| [Insertion sort](http://en.wikipedia.org/wiki/Insertion_sort) | n | n^2 | n^2 | 1 | Yes | Insertion |
| [Selection sort](http://en.wikipedia.org/wiki/Selection_sort) | n^2 | n^2 | n^2 | 1 | No | Selection |
| [Bubble sort](http://en.wikipedia.org/wiki/Bubble_sort) | n | n^2 | n^2 | 1 | Yes | Exchanging |
| [Binary tree sort](http://en.wikipedia.org/wiki/Binary_tree_sort) | n | n \log n | n \log n~\text{(balanced)} | n | Yes | Insertion |

# SORTING

**Non-comparison sorts :**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| [Bucket sort](http://en.wikipedia.org/wiki/Bucket_sort) (uniform keys) | — | n+k | n^2 \cdot k | n \cdot k | Yes | No | Assumes uniform distribution of elements from the domain in the array.[[9]](http://en.wikipedia.org/wiki/Sorting_algorithm#cite_note-clrs-9) |
| [Bucket sort](http://en.wikipedia.org/wiki/Bucket_sort) (integer keys) | — | n+r | n+r | n+r | Yes | Yes | If *r* is O(n), then Average is O(n).[[10]](http://en.wikipedia.org/wiki/Sorting_algorithm#cite_note-gt-10) |
| [Counting sort](http://en.wikipedia.org/wiki/Counting_sort) | — | n+r | n+r | n+r | Yes | Yes | If *r* is O(n), then Average is O(n).[[9]](http://en.wikipedia.org/wiki/Sorting_algorithm#cite_note-clrs-9) |
| [LSD Radix Sort](http://en.wikipedia.org/wiki/Radix_sort#Least_significant_digit_radix_sorts) | — | n \cdot \frac{k}{d} | n \cdot \frac{k}{d} | n + 2^d | Yes | No | [[9]](http://en.wikipedia.org/wiki/Sorting_algorithm#cite_note-clrs-9)[[10]](http://en.wikipedia.org/wiki/Sorting_algorithm#cite_note-gt-10) |
| [MSD Radix Sort](http://en.wikipedia.org/wiki/Radix_sort#Most_significant_digit_radix_sorts) | — | n \cdot \frac{k}{d} | n \cdot \frac{k}{d} | n + 2^d | Yes | No | Stable version uses an external array of size n to hold all of the bins. |

# Eclipse debugging setup C++

To enable STL containers viewing in Eclipse

1. Install GDB 7.5

It has issues in C:\MinGW\share\gdb\python

And

C:\MinGW\share\gdb\syscalls

Take these from gdb-7.5.1\gdb ( download .tar.gz and extract)

1. Install Python 32-bit 2.7 .
2. Set , path and



See the section “ This is for MinGW users with Eclipse CDT” in <http://stackoverflow.com/questions/3651862/better-variable-exploring-when-debugging-c-code-with-eclipse-cdt>

1. Create a file .gdbinit (see above link )

python

import sys

sys.path.insert(0, 'C:/MinGW/share/gcc-4.8.1/python')

from libstdcxx.v6.printers import register\_libstdcxx\_printers

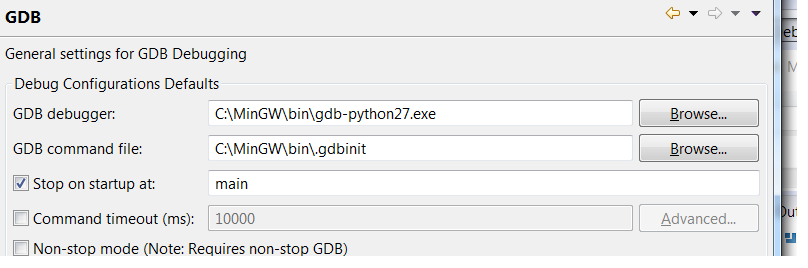
register\_libstdcxx\_printers (None)

end

and copy it to C:\MinGW\bin and then

Set the eclipse debugger settings as follows

1. Note that all projects needs to be rebuild for debugging .



# Difference C++ and java

|  |  |
| --- | --- |
| JAVA | C++ |
| a[] : a is constant , you can’t do a=a+1 | a++ is allowed |
| Can’t do pass by reference (using &) , have to return array of values | Can do pass by reference |
| Can do has next if input size not given | Cannot do input size must be known |
| Can do arr.length ( no brackets ) | Cannot do arr.length  Find size using sizeof(arr)/sizeof(arr[0])  Or , if using std::array then ,  std::array<int,5> myints;  myints.size() << std::endl; |
| Use Integer in place of int , etc to use pass by reference | Can directly pass by reference |
| Can’t do pass by reference , even using Integer aref = new Integer(x)  Since the reference is passed by value and changes to aref will remain local therefore either use  **int**[] minSum = **new** **int**[1];  minSum[0] = val; minimumTotalUtil(minSum );  to pass minsum val as reference  or ,  class MyInteger{  int val;  }  MyInteger aref = new MyInteger()  Aref.val = val  minimumTotalUtil(Aref ){  aref.val = newval;  } | Can do pass by reference |
| JAVA can pass 2D array without giving 2nd dimension  func ( int [][] grid , int row, int col ){  int[][] arr = new int[row][col];  }  func(arr , row , col); | C++ 2nd dimension of array is must or use  int \* visited = new int[row\*col];  memset(visited, 0, row\*col\*sizeof(int));  for ( int i =0; i< row ; i++) {  for ( int j =0; j< col ; j++) {  visited[i\*col+j] = 1;  func(int\* arr){  }  Or ,  Use vector of size (m,n)  vector<vector<int> > arr(m,vector<int>(n,0)); |