

## Contents

1	Basic	
1.1	ascii	
1.2	limits	
2	STL	
2.1	priority_queue	
2.2	map	
2.3	unordered_map	
3	sort	
3.1	big number sort	
3.2	bubble sort	
4	math	
4.1	prime factorization	
5	algorithm	
5.1	basic	
5.2	binarysearch	
6	graph	
6.1	graph	
7	Section2	
7.1	thm	

## 1 Basic

### 1.1 ascii

int	char	int	char	int	char
32		64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	\$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

### 1.2 limits

[Type]	[size]	[range]
char	1	127 to -128
signed char	1	127 to -128
unsigned char	1	0 to 255
short	2	32767 to -32768
int	4	2147483647 to -2147483648
unsigned int	4	0 to 4294967295
long	4	2147483647 to -2147483648

9	unsigned long	4	0 to 18446744073709551615
10	long long	8	
11			9223372036854775807 to -9223372036854775808
12	double	8	1.79769e+308 to 2.22507e-308
13	long double	16	1.18973e+4932 to 3.3621e-4932
14	float	4	3.40282e+38 to 1.17549e-38
15	unsigned long long	8	0 to 18446744073709551615
16	string	32	

## 2 STL

### 2.1 priority\_queue

```

1 priority_queue<int> pq; //宣告
2
3 pq.push(x);
4
5 x = pq.top();
6 pq.pop(); //讀取後刪除
7
8 pq.empty() //回傳 true
9 pq.size() //回傳 0
10
11 priority_queue<T> pq; //預設由大到小
12 priority_queue<T, vector<T>, greater<T>> > pq;
13 //改成由小到大
14 priority_queue<T, vector<T>, cmp> pq; //cmp

```

### 2.2 map

```

1 map：存放 key-value pairs 的映射資料結構，會按 key
  由小到大排序。
2 元素存取
3 operator[]：存取指定的[i]元素的資料
4
5 迭代器
6 begin()：回傳指向map頭部元素的迭代器
7 end()：回傳指向map末尾的迭代器
8 rbegin()：回傳一個指向map尾部的反向迭代器
9 rend()：回傳一個指向map頭部的反向迭代器
10
11 遍歷整個map時，利用iterator操作：
12 取key：it->first 或 (*it).first
13 取value：it->second 或 (*it).second
14
15 容量
16 empty()：檢查容器是否為空，空則回傳 true
17 size()：回傳元素數量
18 max_size()：回傳可以容納的最大元素個數
19
20 修改器
21 clear()：刪除所有元素
22 insert()：插入元素
23 erase()：刪除一個元素
24 swap()：交換兩個map
25
26 查找
27 count()：回傳指定元素出現的次數
28 find()：查找一個元素
29
30 //實作範例
31 #include <bits/stdc++.h>
32 using namespace std;
33
34 int main(){
35
36     //declaration container and iterator
37     map<string, string> mp;
38     map<string, string>::iterator iter;

```

```

39 map<string, string>::reverse_iterator iter_r;
40
41 //insert element
42 mp.insert(pair<string, string>("r000",
43     "student_zero"));
44
45 mp["r123"] = "student_first";
46 mp["r456"] = "student_second";
47
48 //traversal
49 for(iter = mp.begin(); iter != mp.end(); iter++)
50     cout<<iter->first<<" "<<iter->second<<endl;
51 for(iter_r = mp.rbegin(); iter_r != mp.rend();
52     iter_r++)
53     cout<<iter_r->first<<"
54         "<<iter_r->second<<endl;
55
56 //find and erase the element
57 iter = mp.find("r123");
58 mp.erase(iter);
59
60 iter = mp.find("r123");
61
62 if(iter != mp.end())
63     cout<<"Find, the value is
64         "<<iter->second<<endl;
65 else
66     cout<<"Do not Find"<<endl;
67
68 return 0;
69 }
70
71 //map統計數字
72 #include<bits/stdc++.h>
73 using namespace std;
74
75 int main(){
76     ios::sync_with_stdio(0),cin.tie(0);
77     long long n,x;
78     cin>>n;
79     map <int,int> mp;
80     while(n--){
81         cin>>x;
82         ++mp[x];
83     }
84     for(auto i:mp) cout<<i.first<<" "<<i.second<<endl;
85 }

```

## 2.3 unordered\_map

```

1 unordered_map：存放 key-value pairs
  的「無序」映射資料結構。
2 用法與map相同

```

## 3 sort

### 3.1 big number sort

```

1 while True:
2     try:
3         n = int(input())          # 有幾筆數字需要排序
4         arr = []                  # 建立空串列
5         for i in range(n):
6             arr.append(int(input())) # 依序將數字存入串列
7             arr.sort()              # 串列排序
8         for i in arr:
9             print(i)                # 依序印出串列中每個項目
10    except:
11        break

```

## 3.2 bubble sort

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main() {
5     int n;
6     cin>>n;
7     int a[n], tmp;
8     for(int i=0; i<n; i++) cin>>a[i];
9     for(int i=n-1; i>0; i--) {
10         for(int j=0; j<=i-1; j++) {
11             if( a[j]>a[j+1]) {
12                 tmp=a[j];
13                 a[j]=a[j+1];
14                 a[j+1]=tmp;
15             }
16         }
17     }
18     for(int i=0; i<n; i++) cout<<a[i]<<" ";
19 }

```

## 4 math

### 4.1 prime factorization

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main() {
5     int n;
6     while(true) {
7         cin>>n;
8         for(int x=2; x<=n; x++) {
9             while(n%x==0) {
10                 cout<<x<<"*";
11                 n/=x;
12             }
13         }
14         cout<<"\b \n";
15     }
16     system("pause");
17     return 0;
18 }

```

## 5 algorithm

### 5.1 basic

```

1 min：取最小值。
2 min(a, b)
3 min(list)
4 max：取最大值。
5 max(a, b)
6 max(list)
7 min_element：找尋最小元素
8 min_element(first, last)
9 max_element：找尋最大元素
10 max_element(first, last)
11 sort：排序，預設由小排到大。
12 sort(first, last)
13 sort(first, last, comp)：可自行定義比較運算子 Comp。
14 find：尋找元素。
15 find(first, last, val)
16 lower_bound：尋找第一個小於 x
  的元素位置，如果不存在，則回傳 last。
17 lower_bound(first, last, val)
18 upper_bound：尋找第一個大於 x
  的元素位置，如果不存在，則回傳 last。

```

```

19 upper_bound(first, last, val)
20 next_permutation :
    將序列順序轉換成下一個字典序，如果存在回傳 true
    ，反之回傳 false 。
21 next_permutation(first, last)
22 prev_permutation :
    將序列順序轉換成上一個字典序，如果存在回傳 true
    ，反之回傳 false 。
23 prev_permutation(first, last)

```

## 5.2 binarysearch

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int binary_search(vector<int> &nums, int target) {
5     int left=0, right=nums.size()-1;
6     while(left<=right){
7         int mid=(left+right)/2;
8         if (nums[mid]>target) right=mid-1;
9         else if(nums[mid]<target) left=mid+1;
10        else return mid+1;
11    }
12    return 0;
13 }
14
15 int main() {
16     int n, k, x;
17     cin >> n >> k;
18     int a[n];
19     vector<int> v;
20     for(int i=0 ; i<n ; i++){
21         cin >> x;
22         v.push_back(x);
23     }
24     for(int i=0 ; i<k ; i++) cin >> a[i];
25     for(int i=0 ; i<k ; i++){
26         cout << binary_search(v, a[i]) << endl;
27     }
28 }
29
30 /*input
31 5 5
32 1 3 4 7 9
33 3 1 9 7 -2
34 */
35
36 /*output
37 2
38 1
39 5
40 4
41 0
42 */

```

## 6 graph

### 6.1 graph

```

1 #include<bits/stdc++.h>
2 using namespace std;
3
4 class Node {
5 public:
6     int val;
7     vector<Node*> children;
8
9     Node() {}
10
11     Node(int _val) {
12         val = _val;
13     }

```

```

14 }
15
16 Node(int _val, vector<Node*> _children) {
17     val = _val;
18     children = _children;
19 }
20 };
21
22 struct ListNode {
23     int val;
24     ListNode *next;
25     ListNode() : val(0), next(nullptr) {}
26     ListNode(int x) : val(x), next(nullptr) {}
27     ListNode(int x, ListNode *next) : val(x),
28         next(next) {}
29 };
30
31 struct TreeNode {
32     int val;
33     TreeNode *left;
34     TreeNode *right;
35     TreeNode() : val(0), left(nullptr),
36         right(nullptr) {}
37     TreeNode(int x) : val(x), left(nullptr),
38         right(nullptr) {}
39     TreeNode(int x, TreeNode *left, TreeNode *right)
40         : val(x), left(left), right(right) {}
41 };
42
43 class ListProblem {
44     vector<int> nums={};
45 public:
46     void solve() {
47         return;
48     }
49
50     ListNode* buildList(int idx) {
51         if(idx == nums.size()) return NULL;
52         ListNode *current=new
53             ListNode(nums[idx++],current->next);
54         return current;
55     }
56
57     void deleteList(ListNode* root) {
58         if(root == NULL) return;
59         deleteList(root->next);
60         delete root;
61     }
62 };
63
64 class TreeProblem {
65     int null = INT_MIN;
66     vector<int> nums = {}, result;
67 public:
68     void solve() {
69
70         return;
71     }
72
73     TreeNode* buildBinaryTreeUsingDFS(int left, int
74         right) {
75         if((left > right) || (nums[(left+right)/2] ==
76             null)) return NULL;
77         int mid = (left+right)/2;
78         TreeNode* current = new TreeNode(
79             nums[mid],
80             buildBinaryTreeUsingDFS(left,mid-1),
81             buildBinaryTreeUsingDFS(mid+1,right));
82         return current;
83     }
84
85     TreeNode* buildBinaryTreeUsingBFS() {
86         int idx = 0;
87         TreeNode* root = new TreeNode(nums[idx++]);
88         queue<TreeNode*> q;
89         q.push(root);
90     }

```

## 7 Section2

### 7.1 thm

• 中文測試

$$\cdot \sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$

```

84     while(idx < nums.size()) {
85         if(nums[idx] != null) {
86             TreeNode* left = new
87                 TreeNode(nums[idx]);
88             q.front()->left = left;
89             q.push(left);
90         }
91         idx++;
92         if((idx < nums.size()) && (nums[idx] !=
93             null)) {
94             TreeNode* right = new
95                 TreeNode(nums[idx]);
96             q.front()->right = right;
97             q.push(right);
98         }
99         idx++;
100         q.pop();
101     }
102     return root;
103 }
104
105 Node* buildNAryTree() {
106     int idx = 2;
107     Node *root = new Node(nums.front());
108     queue<Node*> q;
109     q.push(root);
110     while(idx < nums.size()) {
111         while((idx < nums.size()) && (nums[idx]
112             != null)) {
113             Node *current = new Node(nums[idx++]);
114             q.front()->children.push_back(current);
115             q.push(current);
116         }
117         idx++;
118         q.pop();
119     }
120     return root;
121 }
122
123 void deleteBinaryTree(TreeNode* root) {
124     if(root->left != NULL)
125         deleteBinaryTree(root->left);
126     if(root->right != NULL)
127         deleteBinaryTree(root->right);
128     delete root;
129     return;
130 }
131
132 void deleteNAryTree(Node* root) {
133     if(root == NULL) return;
134     for(int i=0; i<root->children.size(); i++) {
135         deleteNAryTree(root->children[i]);
136         delete root->children[i];
137     }
138     delete root;
139     return;
140 }
141
142 void inorderTraversal(TreeNode* root) {
143     if(root == NULL) return;
144     inorderTraversal(root->left);
145     cout<<root->val<<' ';
146     inorderTraversal(root->right);
147     return;
148 }
149
150 };
151
152 int main() {
153     return 0;
154 }

```