

## Contents

1	Basic	
1.1	ascii	
1.2	limits	
2	STL	
2.1	priority_queue	
2.2	map	
2.3	unordered_map	
2.4	set	
2.5	multiset	
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	
5.3	prefix sum	
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

6	int	4	2147483647 to -2147483648
7	unsigned int	4	0 to 4294967295
8	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 :
   優先隊列，資料預設由大到小排序，即優先權高的資料會先被取出。
2 宣告：
   priority_queue <int> pq;
4 把元素 x 加進 priority_queue：
   pq.push(x);
5 讀取優先權最高的值：
   x = pq.top();
8   pq.pop();           //讀取後刪除
9 判斷是否為空的priority_queue：
   pq.empty()           //回傳 true
10  pq.size()            //回傳 0
11
12 如需改變priority_queue的優先權定義：
13   priority_queue<T> pq;           //預設由大到小
14   priority_queue<T, vector<T>, greater<T> > pq;
15                                   //改成由小到大
16   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

## 2.4 set

1 set：集合，去除重複的元素，資料由小到大排序。  
 2 宣告：  
 3 set <int> st;  
 4 把元素 x 加進 set：  
 5 st.insert(x);  
 6 檢查元素 x 是否存在 set 中：  
 7 st.count(x);  
 8 刪除元素 x：  
 9 st.erase(x); // 可傳入值或iterator

10 清空集合中的所有元素：  
 11 st.clear();  
 12 取值：使用iterator  
 13 x = \*st.begin(); //  
 set中的第一個元素(最小的元素)。  
 14 x = \*st.rbegin(); //  
 set中的最後一個元素(最大的元素)。  
 15 判斷是否為空的set：  
 16 st.empty() 回傳true  
 17 st.size() 回傳零  
 18 常用來搭配的member function：  
 19 st.count(x);  
 20 auto it = st.find(x); // binary search, O(log(N))  
 21 auto it = st.lower\_bound(x); // binary search,  
 O(log(N))  
 22 auto it = st.upper\_bound(x); // binary search,  
 O(log(N))  
 23 【multiset】  
 24 與 set  
 用法雷同，但會保留重複的元素，資料由小到大排序。  
 25 宣告：  
 26 multiset<int> st;  
 27 刪除資料：  
 28 st.erase(val); 會刪除所有值為 val 的元素。  
 29 st.erase(st.find(val)); 只刪除第一個值為 val 的元素。

## 2.5 multiset

1 與 set 用法雷同，但會保留重複的元素，  
 資料由小到大排序。  
 2 宣告：  
 3 multiset<int> st;  
 4 刪除資料：  
 5 st.erase(val); 會刪除所有值為 val 的元素。  
 6 st.erase(st.find(val)); 只刪除第一個值為 val  
 的元素。

## 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 /*
31 input
32 5 5
33 1 3 4 7 9
34 3 1 9 7 -2
35 */
36
37 /*
38 output
39 2
40 1
41 5
42 4
43 0
44 */

```

### 5.3 prefix sum

```

1 // 前綴和
2 // 陣列前n項的和。
3 // b[i] = a[0] + a[1] + a[2] + ... + a[i]
4 // 區間和 [l, r]：b[r]-b[l-1] (要保留b[l]所以-1)
5
6 #include <bits/stdc++.h>
7 using namespace std;
8 int main(){
9     int n;
10    cin >> n;
11    int a[n], b[n];
12    for(int i=0; i<n; i++) cin >> a[i];
13    b[0] = a[0];
14    for(int i=1; i<n; i++) b[i] = b[i-1] + a[i];
15    for(int i=0; i<n; i++) cout<<b[i]<<' ';
16    cout<<"\n";
17    int l, r;
18    cin >> l >> r;
19    cout << b[r] - b[l-1] ; //區間和
20 }
21
22
23 // 差分
24 // 用途：在區間 [l, r] 加上一個數字v。
25 // b[l] += v; //b[0~l] 加上v
26 // b[r+1] -= v; //b[r+1~n] 減去v (b[r] 仍保留v)
27 // 給的 a[] 是前綴和數列，建構 b[]，
28 // 因為 a[i] = b[0] + b[1] + b[2] + ... + b[i]，

```

```

29 // 所以 b[i] = a[i] - a[i-1]。
30 // 在 b[l] 加上 v，b[r+1] 減去 v，
31 // 最後再從 0 跑到 n 使 b[i] += b[i-1]。
32 // 這樣一來，b[] 是一個在某區間加上v的前綴和。
33
34 #include <bits/stdc++.h>
35 using namespace std;
36 int a[1000], b[1000];
37 //a: 前綴和數列, b: 差分數列
38 int main(){
39     int n, l, r, v;
40     cin >> n;
41     for(int i=1; i<=n; i++){
42         cin >> a[i];
43         b[i] = a[i] - a[i-1]; //建構差分數列
44     }
45     cin >> l >> r >> v;
46     b[l] += v;
47     b[r+1] -= v;
48
49     for(int i=1; i<=n; i++){
50         b[i] += b[i-1];
51         cout << b[i] << ' ';
52     }
53 }

```

## 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     Node(int _val, vector<Node*> _children) {
16         val = _val;
17         children = _children;
18     }
19 };
20
21 struct ListNode {
22     int val;
23     ListNode *next;
24     ListNode() : val(0), next(nullptr) {}
25     ListNode(int x) : val(x), next(nullptr) {}
26     ListNode(int x, ListNode *next) : val(x),
27         next(next) {}
28 };
29
30 struct TreeNode {
31     int val;
32     TreeNode *left;
33     TreeNode *right;
34     TreeNode() : val(0), left(nullptr),
35         right(nullptr) {}
36     TreeNode(int x) : val(x), left(nullptr),
37         right(nullptr) {}
38     TreeNode(int x, TreeNode *left, TreeNode *right)
39         : val(x), left(left), right(right) {}
40 };
41
42 class ListProblem {
43     vector<int> nums={};
44 };

```

```

41 public:
42     void solve() {
43         return;
44     }
45
46     ListNode* buildList(int idx) {
47         if(idx == nums.size()) return NULL;
48         ListNode *current=new
49             ListNode(nums[idx++],current->next);
50         return current;
51     }
52
53     void deleteList(ListNode* root) {
54         if(root == NULL) return;
55         deleteList(root->next);
56         delete root;
57         return;
58     }
59 };
60
61 class TreeProblem {
62     int null = INT_MIN;
63     vector<int> nums = {}, result;
64 public:
65     void solve() {
66         return;
67     }
68
69     TreeNode* buildBinaryTreeUsingDFS(int left, int
70         right) {
71         if((left > right) || (nums[(left+right)/2] ==
72             null)) return NULL;
73         int mid = (left+right)/2;
74         TreeNode* current = new TreeNode(
75             nums[mid],
76             buildBinaryTreeUsingDFS(left,mid-1),
77             buildBinaryTreeUsingDFS(mid+1,right));
78         return current;
79     }
80
81     TreeNode* buildBinaryTreeUsingBFS() {
82         int idx = 0;
83         TreeNode* root = new TreeNode(nums[idx++]);
84         queue<TreeNode*> q;
85         q.push(root);
86         while(idx < nums.size()) {
87             if(nums[idx] != null) {
88                 TreeNode* left = new
89                     TreeNode(nums[idx]);
90                 q.front()->left = left;
91                 q.push(left);
92             }
93             idx++;
94             if((idx < nums.size()) && (nums[idx] !=
95                 null)) {
96                 TreeNode* right = new
97                     TreeNode(nums[idx]);
98                 q.front()->right = right;
99                 q.push(right);
100             }
101             idx++;
102             q.pop();
103         }
104         return root;
105     }
106
107     Node* buildNaryTree() {
108         int idx = 2;
109         Node *root = new Node(nums.front());
110         queue<Node*> q;
111         q.push(root);
112         while(idx < nums.size()) {
113             while((idx < nums.size()) && (nums[idx]
114                 != null)) {
115                 Node *current = new Node(nums[idx++]);
116                 q.front()->children.push_back(current);
117             }
118             q.pop();
119             idx++;
120         }
121     }

```

```

111         q.push(current);
112     }
113     idx++;
114     q.pop();
115 }
116 return root;
117 }
118
119 void deleteBinaryTree(TreeNode* root) {
120     if(root->left != NULL)
121         deleteBinaryTree(root->left);
122     if(root->right != NULL)
123         deleteBinaryTree(root->right);
124     delete root;
125     return;
126 }
127
128 void deleteNaryTree(Node* root) {
129     if(root == NULL) return;
130     for(int i=0; i<root->children.size(); i++) {
131         deleteNaryTree(root->children[i]);
132         delete root->children[i];
133     }
134     delete root;
135     return;
136 }
137
138 void inorderTraversal(TreeNode* root) {
139     if(root == NULL) return;
140     inorderTraversal(root->left);
141     cout<<root->val<< ' ';
142     inorderTraversal(root->right);
143     return;
144 }
145 };
146
147 int main() {
148     return 0;
149 }

```

## 7 Section2

### 7.1 thm

- 中文測試

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