insertion sort

앞 value랑 계속 비교해서 정렬해 나가는 정렬

#include <iostream>  
using namespace std;  
​  
int main(){  
   int arr[6] = {4,9,11,8,6,2};  
   int result[6];//내림차순 정렬  
     
   for(int i=0;i<6;i++){  
       result[i]=arr[i];  
         
       for(int j=i;j>0;j--){  
           if(result[j-1] < result[j]){  
               swap(result[j-1],result[j]);  
          }  
           else{  
               break;  
          }  
      }  
  }  
     
   return 0;  
}

숫자 오름차순, 문자 내림차순으로 정렬

#include<iostream>  
#include<vector>  
using namespace std;  
struct node {  
int first;  
char second;  
};  
node arr[5]{  
{1,'A'},  
{3,'A'},  
{2,'C'},  
{1,'B'},  
{3,'B'},  
};  
node result[5];  
​  
bool compare(node front, node back) {  
if (front.first < back.first) return true; // 앞의 숫자가 더 작아야 한다.  
if (front.first > back.first) return false;  
return front.second > back.second;  
}  
​  
int main() {  
  
​  
for (int i = 0; i < 5; i++) {  
result[i] = arr[i];  
​  
for (int j = i; j >= 0; j--) {  
node front = result[j - 1];  
node back = result[j];  
​  
if (!compare(front, back)) {  
swap(result[j - 1], result[j]);  
}  
else  
break;  
}  
}  
​  
for (int i = 0; i < 5; i++) {  
cout << result[i].first<<" "<<result[i].second << " ";  
}  
cout << endl;  
​  
return 0;  
}

sort -> quick, heap sort

=> STL sort

일반적인 STL sort 함수는 quick이나 heap sort의 속도로 동작되기 때문에 빠르다

O(nlogn) 속도

또한 위의 예시와 같이 compare함수를 정의해 주고 sort함수의 뒤에 써주면

Compare에 명시된 차순으로 정렬된다.

#include <iostream>  
#include <vector>  
#include <string>  
#include <algorithm>  
using namespace std;  
​  
​  
bool compare(char front, char back) {  
​  
int aa = 0, bb = 0;  
if (front >= 'H' && front <= 'O') aa = 1;  
if (back >= 'H' && back <= 'O') bb=1;  
​  
if (aa == 1 && bb == 0) return true;  
if (aa == 0 && bb == 1)return false;  
​  
return front > back;  
}  
​  
int main() {  
​  
char arr[16] = "KEVINCHOIGOODMAN";  
​  
sort(str, str+16, compare);  
​  
cout << str << endl;  
​  
return 0;  
}

count sort

1. DAT
2. 누적합
3. 값 넣기

str => O(nlogn)

count sort => O(n)

#include <iostream>  
#include <vector>  
#include <string>  
#include <algorithm>  
using namespace std;  
​  
int main() {  
​  
int bucket[11] = {};  
int arr[7] = { 3,9,1,2,1,3,4 };  
​  
for (int i = 0; i < 7; i++) {  
bucket[arr[i]]++;  
}  
​  
for (int i = 1; i < 11; i++) {  
bucket[i] += bucket[i - 1];  
}  
​  
int result[11] = {};  
​  
for (int i = 0; i < 7; i++) {  
result[bucket[arr[i]]--] = arr[i];  
}  
for (int i = 1; i < 11; i++)  
cout << result[i] << " ";  
cout << endl;  
​  
return 0;  
}

Dat에 나온 숫자를 index별로 기록해 두고 직전 값과 더한 값 즉 누적합을 계산해 준 뒤, 앞에서부터 하나씩 빼주면서 정렬해주는 count sort

union - find

-> 각각 독립된 data를 그룹화 시켜서 관리할 때 사용

#include <iostream>  
​  
using namespace std;  
​  
char arr[200];  
​  
char findboss(char member) {  
if (arr[member] == 0) {  
return member; //자기가 보스일 때  
}  
char boss = findboss(arr[member]);  
return boss;  
}  
​  
void setunion(char a, char b) {  
​  
char aboss = findboss(a);  
char bboss = findboss(b);  
​  
if (aboss == bboss) return;  
arr[bboss] = aboss;  
​  
}  
​  
int main()  
{  
setunion('A', 'B');  
setunion('E', 'F');  
setunion('B', 'F');  
setunion('F', 'A');  
setunion('C', 'D');  
​  
char input1, input2;  
cin >> input1 >> input2;  
​  
if (findboss(input1) == findboss(input2)) {  
cout << "같은 그룹입니다" << endl;  
}  
else {  
cout << "다른 그룹입니다" << endl;  
}  
​  
return 0;  
​  
}

Union find 알고리즘은 A,B가 있을 때 하나를 하나의 자식으로 설정해 두어 찾고자 하는 문자나 숫자의 부모 노드를 찾는 그룹화 알고리즘이다.

setunion함수와 findboss 함수를 이용해 설정할 수 있다.

크루스칼 알고리즘

=> 신장트리에서 최소값 찾을 때 사용

#include <iostream>  
#include<vector>  
#include<algorithm>  
using namespace std;  
​  
int arr[200];  
​  
char findboss(char member) {  
if (arr[member] == 0) {  
return member; //자기가 보스일 때  
}  
char boss = findboss(arr[member]);  
return boss;  
}  
​  
void setunion(char a, char b) {  
​  
char aboss = findboss(a);  
char bboss = findboss(b);  
​  
if (aboss == bboss) return;  
arr[bboss] = aboss;  
​  
}  
struct node {  
char a;  
char b;  
int c;  
};  
bool compare(node a, node b) {  
if (a.c > b.c) return false;  
if (a.c < b.c)return true;  
return false;  
}  
int main()  
{  
int sz;  
cin >> sz;  
int n;  
cin >> n;  
int cnt = 0;  
int sum = 0;  
vector<node> arr;  
for (int i = 0; i < n; i++) {  
node temp;  
cin >> temp.a >> temp.b >> temp.c;  
arr.push\_back(temp);  
}  
sort(arr.begin(), arr.end(),compare);  
for (int i = 0; i < n; i++) {  
if (cnt == sz - 1)  
break;  
if (findboss(arr[i].a) == findboss(arr[i].b)) {  
continue;  
}  
setunion(arr[i].a, arr[i].b);  
sum += arr[i].c;  
cnt++;  
}  
​  
cout << sum << endl;  
​  
return 0;  
​  
}

크루스칼 알고리즘은 union find 알고리즘을 이용해 짤 수 있는데 struct를 이용해서 노드 사이의 비용을 저장하고 그룹화 하면서 최소 비용의 노드를 잇는 알고리즘이다.

BS(binary search)

=> 이미 정렬이 되어 있는 배열에서 내가 원하는 값을 찾고자 할 때 이용

#include <vector>  
#include <iostream>  
#include <algorithm>  
using namespace std;  
int target;  
vector<int> arr={1,2,5,33,44,63,34,99};  
​  
int binarySearch(int start, int end, int target){  
   while(1){  
       int mid = (start+end)/2;  
       if(vect[mid]==target) return 1;  
       if(start>end) return -1;  
       if(target>vect[mid])start=mid+1;  
       else end = mid-1;  
  }  
}  
​  
int main(){  
cin>>target;  
     
   int ret=binarySearch(0,arr.size()-1,target);  
   if(ret==1) cout<<target<<"찾음"<<endl;  
   else cout<<target<<"못 찾음"<<endl;  
     
   return 0;  
}

Target 값을 빠른 속도로 찾을 때 이용하는 알고리즘으로 정렬이 되어 있는 배열을 사용해서 값을 찾을 때 사용한다.

Start는 0번, end는 배열의 마지막 , mid는 배열의 중간 인덱스로 설정하고 target 값을 mid 값과 비교한 후 mid 값보다 작으면 end 인덱스를 mid-1로 설정, 크면 start 값을 mid+1로 설정하면서 반복한다. 만약 mid값이 target값과 같으면 find, start 인덱스가 end 인덱스보다 커질 경우 not find 처리를 해준다.

parameteric search

#include <iostream>  
#include <string>  
#include <vector>  
#include <algorithm>  
using namespace std;  
​  
string battery;  
int max\_idx = -1;  
int BS(int start, int end, char target) {  
​  
while (1) {  
  
int mid = (start + end) / 2;  
if (start > end) break;  
if (battery[mid] == target) {  
if (max\_idx < mid)  
max\_idx = mid;  
start = mid + 1;  
}  
else end = mid - 1;  
  
}  
return max\_idx;  
}  
​  
int main() {  
​  
cin >> battery;  
​  
int ret = BS(0, battery.length() - 1, '\*');  
​  
cout << ((ret+1)) \* 10 << "%" << endl;  
​  
return 0;  
}

=> 해당 target이 어느 정도 있는지 찾는 알고리즘으로 max\_idx라는 변수에 현재 mid에 target이 있을 경우 그 위치를 저장하면서 계속 비교해 가는 이진 검색 알고리즘이다.