자료구조 기초 공부 – 기초 코드 작성 요령

강의출처: <https://blog.encrypted.gg/922?category=773649>

텍스트이(가) 표시된 사진

자동 생성된 설명

2번째 줄에서 cnt=0으로 선언할 때 1번

I 를 초기값으로 0 줄때 1번

{I가 n보다 작은지 확인하고 i를 1증가시키면서 연산횟수 2번

Arr[i]값을 나누기 연산하고 0인지 비교할 때 연산횟수 2번

만약 같을 때 cnt 1증가 할 때 1번}

마지막으로 cnt 반환할때 1번

이렇게 하면 너무 노가다

**N에 비례한다! 라고 생각하면 간단**

텍스트이(가) 표시된 사진

자동 생성된 설명

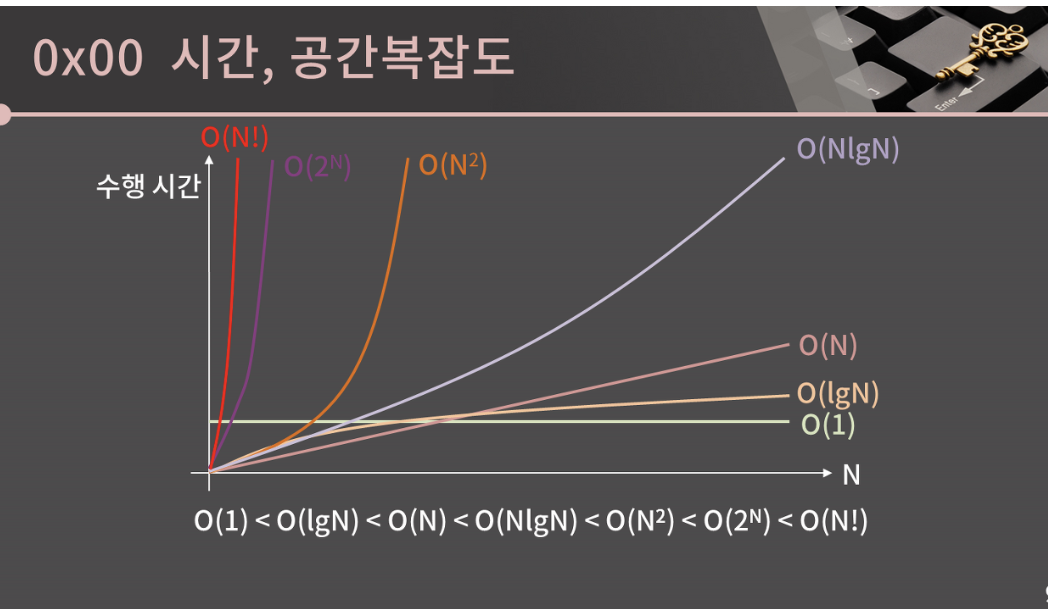
텍스트이(가) 표시된 사진

자동 생성된 설명

lg 2 = 1, lg 4 = 2, lg 8 = 3, lg 16 = 4, lg 32 = 5 뭐 이정도만 이해하고 가셔도 당장 이해하는데는 큰 문제가 없습니다.  이정도로만 생각해도 문제없음

텍스트이(가) 표시된 사진

자동 생성된 설명



테이블이(가) 표시된 사진

자동 생성된 설명

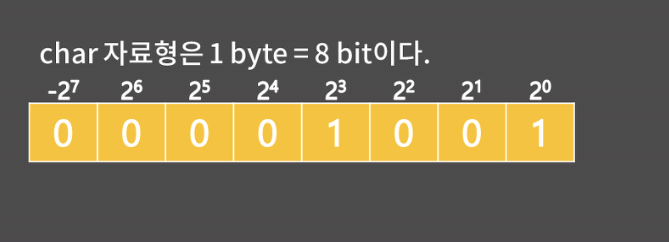
주어진 문제를 보고 풀이를 떠올린 후에 무턱대고 바로 그걸 짜는게 아니라 **내 풀이가 이 문제를 제한 시간 내로 통과할 수 있는지, 즉 내 알고리즘의 시간복잡도가 올바른지를 꼭 생각해봐야 합니다.**

공간복잡도

메모리 제한이 512MB일 때 int변수를 대략 1.2억개 사용가능

Int 1개가 4byte

Char의 자료형은 1byte = 8bit이다.





unsigned char로 표현할 수 있는 수의 최솟값과 최댓값을 고민해보면 최솟값은 00000000일 때 0일거고, 최댓값은 11111111일 때 255입니다.

char에 대해서도 똑같이 표현할 수 있는 최솟값과 최댓값이 얼마일지 생각해보면 최솟값은 10000000일 때 -128일거고, 최댓값은 01111111일 때 127입니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

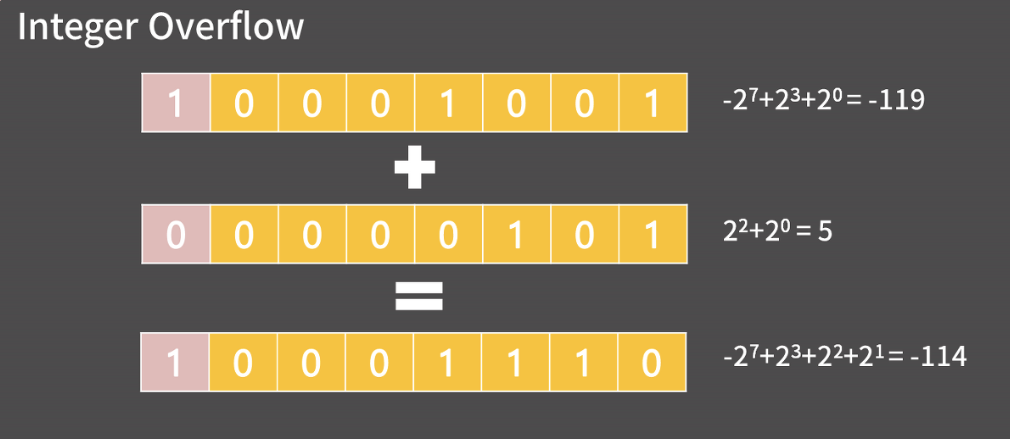
정수 int가 표현할수 있는 최대 정소 21억

Short는 쓸일이 없고 80번째 피보나치수를 구하는 문제와 같이 int 자료형을 쓸수 있는 범위를 넘어서는 수를 저장해야한다면 반드시 **long long 자료형을** 사용해야 한다.

**Int는 10^9까지** 쓸 수 있다고 생각하자

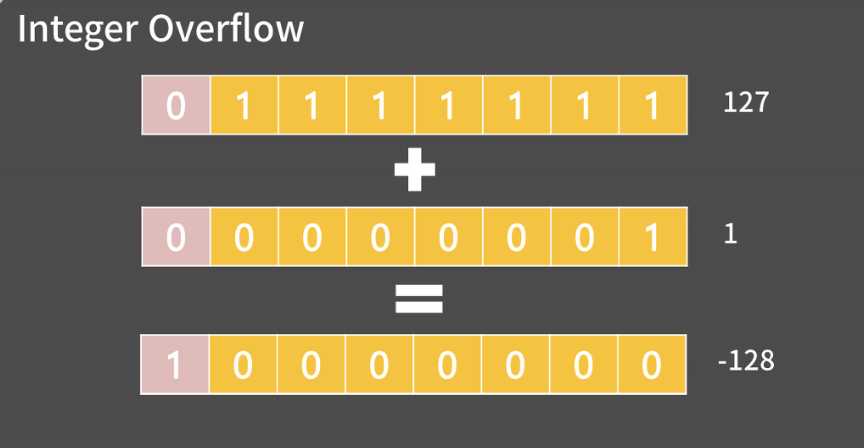
만약 unsigned long long의 범위를 넘어서는 수를 저장하는 것을 요구한다면 string을 활용하여 저장해야한다.

연산을 해야한다면 그냥 python을 쓰는게 났다.



음수 섞인 연산이더라도 정상적으로 수행

인티저 오버플로우란 다음과 같다.



127+1=-128이라는 결과를 볼 수 있다. 이것이 인티저 오버플로우이다!

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

fraction field를 가지고 각 자료형이 어디까지 정확하게 표현할 수 있는지 보면 float은 유효숫자가 6자리이고 double은 유효숫자가 15자리입니다. 이 말은 곧 float은 상대 오차 10-6까지 안전하고 double은 10-15까지 안전하다는 소리입니다.

상대 오차가 10-15까지 안전하다는 표현을 정확하게 이해할 필요가 있는데, **원래 참값이 1이라고 할 때, 1-10-15 에서 1+10-15 사이의 값을 가진다는게 보장된다**는 의미입니다. 즉 오차가 생기는 것 자체는 막을 수가 없지만 오차가 어느 정도인지는 알 수 있습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

두 번째로, double에 long long 범위의 정수를 함부로 담으면 안됩니다. **double은 유효숫자가 15자리인데 long long은 최대 19자리**니까 1018+1과 1018을 구분할 수가 없고 그냥 같은 값이 저장됩니다. 즉, double에 long long 범위의 정수를 담을 경우 오차가 섞인 값이 저장될 수 있습니다.

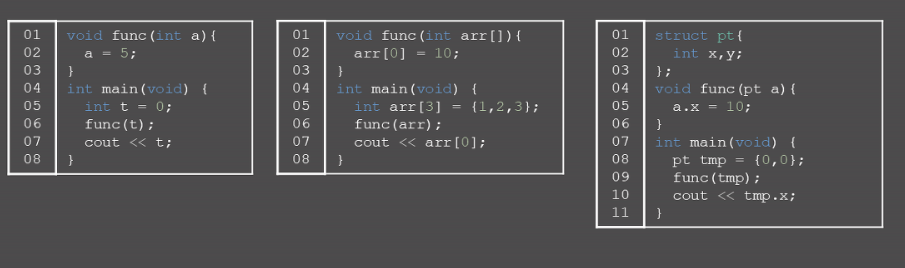
다만 int는 최대 21억이기 때문에 double에 담아도 오차가 생기지 않습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

오차 때문에 두 실수가 같은지 알고 싶을 때에는 둘의 차이가 아주 작은 값, **대략 10-12 이하면 동일하다고 처리를 하는게 안전합니다.**

5번째 줄의 1e-12가 처음 보는 표현일 수 있을 것 같은데 저게 바로 10-12입니다. 비슷하게 만약 109 가 필요하면 1000000000이라고 써도 되긴 한데 아무래도 이렇게 쓰면 0 갯수를 세는 것도 힘들고 하니까 대신에 1e9라고 써도 됩니다.

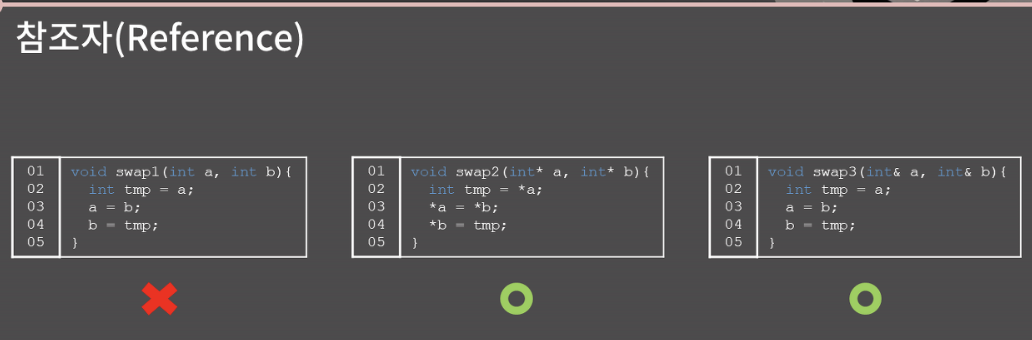


정답은 0, 10, 0

Int는 값이 복사되어 가고

배열은 주소를 전달하고

구조체도 마찬가지 그냥 값이 복사되기 때문에 다음과 같은 결과가 나옴



그래서 swap2 함수처럼 포인터를 보내서 두 변수의 값을 바꿀 수가 있습니다. 그런데 C++에서는 해결법이 한 개 더 있는데, 바로 참조자(reference)입니다. swap3 함수를 보면 함수 인자인 a와 b의 type이 int가 아니고, int 뒤에 &가 붙어있는 것을 볼 수 있습니다. 그러니까 a와 b는 int reference인 것입니다.

저렇게 a와 b를 참조자로 만들면 함수 내의 코드에서는 **그냥 int를 쓰듯이 tmp에 a를 대입하고, a에 b를 대입하고 하는데 저게 다 원본을 바꾸는 것입니다.**

텍스트이(가) 표시된 사진

자동 생성된 설명

여기서 답은 뭘까

바로 0 이다.

STL도 구조체랑 비슷하게 함수 인자를 실어보내면 복사본을 보냄

Func1함수에서 바꾼건 원본에 영향주지 않음

텍스트이(가) 표시된 사진

자동 생성된 설명

그냥 STL을 쌩으로 함수 인자에 넣으면 복사해서 보낸다는걸 유의해야함

두벡터를 인자로 넘겨받아 idx번째 원소의 값을 비교한 결과를 반환하는 함수

여기의 시간복잡도 충격적이게도 O(N)

아니 함수 연산을 딱 1번하는데 어케 O(N)이지??

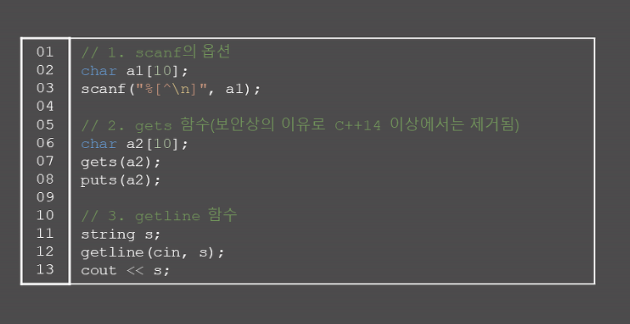
V1,v2 인자를 보낼 때 원본으로부터 복사본을 만드는 비용을 생각하지 못한 것

V1,v2의 크기가 N이니까 N개의 원소들을 하나하나 복사하는 과정은 O(N)이 든다.

텍스트이(가) 표시된 사진

자동 생성된 설명

cmp2 함수에서는 v1, v2의 type을 vector의 reference로 만들었습니다. 그러면 cmp2가 호출될 때 복사본을 따로 만들어내지 않고 참조 대상의 주소 정보만 넘어가기 때문에 시간복잡도는 의도한대로 O(1)이 됩니다.



공백포함해서 문자열 받는 방법

단순히 cin을 사용하면 안된다.

Cin과 getline을 함께사용한다면 cin 이후에 cin.ignore()해야 뒤에 개행문자가 삭제되고 정상적으로 cin.ignore를 할 수 있다.

cin.ignore()

텍스트이(가) 표시된 사진

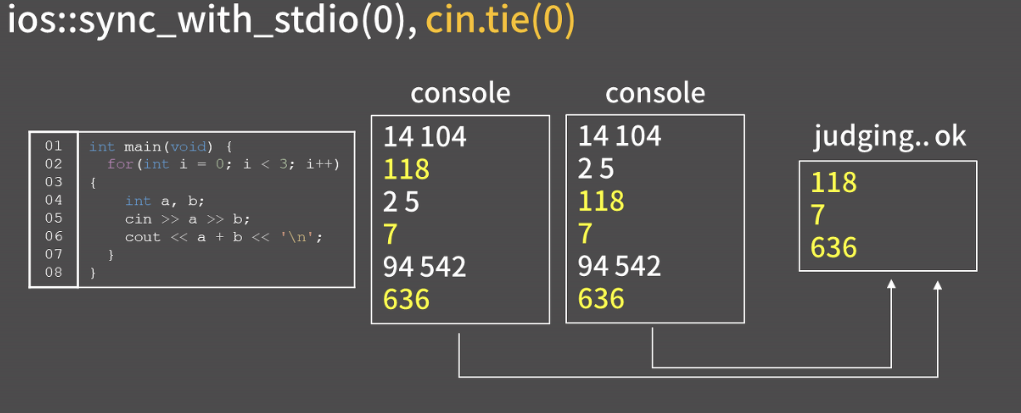
자동 생성된 설명  
그렇기 때문에 C++ stream만 쓸거면 동기화를 끊어버려서 프로그램 수행 시간에서 이득을 챙길 수 있고, 동기화를 끊는 명령이 sync\_with\_stdio(0)입니다. 엄밀히 말해 인자가 bool type이라 sync\_with\_stdio(false)가 더 맞긴 한데 false보다 0이 더 짧으니 그냥 0으로 하겠습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그렇지 않고 출력 버퍼라는 곳에 문자가 임시로 저장되었다가 버퍼가 비워지면서 화면에 보입니다.

출력에서 버퍼가 있는 것 처럼, 입력에서도 버퍼가 있어서 키보드로 받은 입력을 바로바로 넘겨주지 않고 버퍼에서 어느 정도 모았다가 줍니다.



그런데 입력과 출력이 번갈아나오고 그게 한 화면에서 다 보여질 경우에는 버퍼의 존재로 인해서 순서가 꼬여버릴 수도 있습니다.

지금 이 코드를 보면 3번에 걸쳐 두 수를 입력받고 합을 출력하는데 우리는 당연히 순서에 맞게 콘솔에 나타나기를 원합니다. 그런데 예를 들어 6번 줄에서 "118\n"이란 출력을 하라고 했을 때, 바로 출력이 되지 않고 버퍼에 들어있다가 "2 5"를 입력한 후에야 출력이 되면 순서가 꼬일 것입니다.

이런 현상을 막으려고 기본적으로는 cin 명령을 수행하기 전에 cout 버퍼를 비워줍니다. 버퍼를 비우면 자연스럽게 "2 5" 입력이 들어오기 전에 118이 출력될거고 또 "94 542" 입력이 들어오기 전에 7이 출력되어서 순서가 꼬이지 않게 됩니다.

그런데 온라인 저지 사이트에서는 채점을 할 때 그냥 출력 글자만 확인을 합니다. 그렇기 때문에 콘솔 창에서 입력 글자와 출력 글자 사이에 순서가 설령 꼬인다고 해도 채점에 아무런 영향을 주지 않고 두 경우 모두 다 정답 처리가 됩니다.

그러면 굳이 cin 명령을 수행하기 전에 cout 버퍼를 비울 필요가 없다는걸 알 수 있습니다. **그래서 cin 명령을 수행하기 전에 cout 버퍼를 비우지 않도록 하는 코드가 cin.tie(nullptr)**인거고, 엄밀히는 type을 지켜서 nullptr로 쓰는게 좋지만 그냥 타이핑도 아낄겸 0으로 쓰겠습니다.



endl은 개행문자("\n")를 출력하고 출력 버퍼를 비워라는 명령입니다. 앞에서도 얘기했지만 어차피 저지는 프로그램이 종료될 때 출력이 어떻게 생겼는지를 가지고 채점을 진행하니까 중간 중간 버퍼를 비우라고 명령을 줄 필요가 전혀 없습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명



텍스트이(가) 표시된 사진

자동 생성된 설명

대충 왼쪽 코드와 같은 방식인데, 일단 필요한 헤더만 include 했고, 딱 크기에 맞게 n칸짜리 배열을 잡은 후에 해제까지 합니다. 아주 깔끔하죠.

그런데 코딩테스트에서는 내가 헷갈리지 않는 범위 안에서 어떻게든 타이핑을 아끼는게 최고입니다. 그래서 저는 오른쪽 위와 같이 짤 것입니다.