프로그래밍 언어 과제1

20193010 심상현

수행 작업

해당 과제에서의 주요한 작업은 입력받기, 정렬, GCD 계산, 소수 개수 구하기입니다.

입력 받기

C++ / Java

C++와 Java의 입력은 매우 유사해서 하나의 글로 작성합니다.

해당 입력은 모두 while(1) / while(true) 내에서 돌아가서, 혹시라도 잘못된 값을 입력 받아올 경우, while문만 다시 호출하는 것으로 활용하였습니다. 단, C++의 경우 `ios::sync_with_stdio(false);`코드를 이용해서 cpp의 iostream과 c의 stdio와의 동기화를 끊어주어 실행속도를 높여야합니다.

```
ios::sync_with_stdio(false);
```

다음 데이터를 저장하기 위해서 자료구조 중 `map`을 사용하였습니다. map 자료구조는 key와 value로 데이터가 나뉘어지며, key값은 중복되지 않습니다. 여기서 key 값이 중복되지 않는 특성을 이용하여, 데이터가 중복되지 않도록 하였습니다.

그리고 입력 받는 데이터의 크기가 100,000을 벗어난, 너무나 큰 값으로 입력 받을 시에는 continue를 하여 처음 데이터를 입력받는 while()로 돌아가게 됩니다. 여기서 입력 받는 데이터의 크기를 생각하여 long long 형태로 크기를 정하여 충분히 큰 값이 들어오더라도 어느 정도는 입력을 받을 수 있도록 하였습니다.

그 뒤에 입력된 데이터들이 모두 동일하여 실제로 들어온 데이터의 수가 1인 경우를 인지하여, map의 size가 1인 경우 이또한 다시 while()로 돌아가도록 설정하였습니다.

```cpp
flag = 0;
cout << "Input Size : ";
cin >> N;
if(N >= 31 || N <= 1) {
    cout << "your input data is too big or too small\n";
    continue;
}
cout << "Input Numbers : ";

map <int, int> Map;
for(int i=0; i<N; i++) {
    cin >> temp;
    if(temp >= MAX) {
        flag = 1;
        break;
    }
    Map[temp] = 1;
}

if(flag == 1) {
    cout << "your input data is too big!\n";
    continue;
}
else if(Map.size() == 1) {
    cout << "your input data is overlapped, so actual number of input data is 1\n";
    continue;
}
```

C++의 input 영역. 입력 받은 데이터를 Map의 key값으로 넣어줌으로, 중복을 제거해준다.

Python

python은 `input()`을 하면 해당 데이터를 str형태로 받아옵니다. 이를 int형으로 형변환한다면, int형으로 바뀌고, 이것이 intger형의 최댓값을 넘어가면 자동으로 long형으로 변환되고, 이것의 최댓값은 사실상 존재하지 않음으로, 매우 큰 입력을 받을 수 있습니다.

그 뒤 해당 데이터들만큼 input을 split하여 int형으로 mapping하여 arr(list)에 넣어줍니다. 그리고 이것 list의 값들을 dictionary의 key값들로 넣어줍니다. dictionary는 다른 언어들의 자료구조 map과 동일한 형태로, key값의 중복을 없애주는 역할을 합니다. 그리고 다시 이것을 list형태로 바꿔주고, 각 값들이 100,000보다 커지면 `while(True)`로 돌아가게 해줍니다.

```python
while(True) :
    print("plz input your number")
    INPUT = int(input())
    if INPUT >= 31 or INPUT <= 1 :
        print("your input is too big or small")
        continue
    print("input numbers for calculation")
    arr = list(map(int, input().split()))
    start = time.time()
    tempDic = {int : 0 for int in arr}
    arr = list(tempDic.keys())


    flag = False
    for i in arr :
        if i > 100000 :
            flag = True
    if flag == True or len(arr) == 1 :
        print("your input numbers is too big or not overlapped data size is 1")
        continue
```

Python의 input 영역. 입력 받은 list를 dictionary로 바꾸었다가 다시 list로 변환시킴으로 중복된 값들을 모두 없애버린다.

```
plz input your number
3
input numbers for calculation
18189156159498456156 123 456
your input numbers is too big or not overlapped data size is 1
plz input your number
2
```

input data의 값이 매우 큰 경우. 입력 값이 일반적인 입력 값을 초과하더라도, 객체 자체적으로 값들을 크기 비교를 할 수 있음으로, 따로 제한선을 두지 않아도 괜찮다.

정렬

정렬하는 법은 각 언어의 내장 함수인 sort를 사용하였습니다. 각각의 제공되는 sorting은 하나의 sort 알고리즘만을 사용하는 것이 아니라 여러가지 알고리즘들을 결합하여 사용하는 hybrid sorting임으로, 단일 sorting 알고리즘을 사용하는 것보다 시간 복잡도가 획기적으로 줄어듭니다.
단, 본 과제에서의 C++는 sorting 함수 자체를 사용하지 않았습니다.

C++

C++의 map 데이터 구조는 각 key값을 정렬하여 입력받아 주게됩니다. Map의 시작지점을 가리키는 변수 iter를 이용하여, 존재하는 모든 key값들을 찾아주고 그 값들을 `arr[]`에 입력해주었습니다. 이것은 애초에 map 자료구조에서 정렬되어 데이터가 쌓이기 때문에, arr[]는 자동으로 정렬된 값들이 넣어지게 됩니다. 교수님께서 최대한 동일한 알고리즘으로 구현하는 것을 요청하셨지만, 이미 정렬되어 있는 값을 다시 정렬하는 것은 그저 시간/공간 복잡도를 높이는 일이라고 생각하여 제외하였습니다.

만약 key 값들이 정렬되어 있지 않았다면, C++ STL 내장함수 `sort()`를 사용하였을 것입니다. 이 함수는 introsort 알고리즘을 사용합니다. 이는 quick sort, heap sort, insertion sort를 복합하여 quick sort 최악 시간복잡도를 보일 가능성이 있을 때, heap sort와 insertion sort를 조합하여 평균 시간복잡도를 O(n log n)으로 만드는 sorting입니다.

```
arr.resize(Map.size(), 0);
int i=0;
for(map<int,int>::iterator it = Map.begin(); it != Map.end(); it++) {
    arr[i++] = it->first;
}
```

C++의 데이터. 이미 map형식으로 key값들이 sort가 된 데이터들이 되어, it변수로 Map의 값들을 하나씩 가르키면서 해당 데이터들을 arr[]에 넣어주었다.

```
sort(arr, arr+Map.size());
```

Java

Java의 경우 기본형 타입의 배열인 경우 DualPivotQuicksort를 사용하고, reference 타입의 배열인 경우 Tim sort를 사용합니다. 저의 경우에는 `Collecitons.sort()` 함수를 사용하였는데, 이는 내부적으로 List 인터페이스의 sort() 메소드를 실행합니다. 따라서 Tim sort가 실행되는 것으로 알 수 있습니다. Tim sort의 이론은 아래 python에서 설명합니다.

```
Collections.sort(data);
```

Python

Python의 경우 내장 sort함수는 Tim sort로, 삽입 정렬과 병합 정렬을 함께 사용합니다. 병합 정렬의 단점인 추가적인 메모리가 필요하다는 것을 삽입 정렬을 결합함으로서 추가 메모리를 적게 잡아서 사용할 수 있습니다.

```
arr.sort()
```

GCD 계산

GCD 계산은 유클리드 호제법을 이용하여 해결하였습니다.

sort된 array에 index 0와 index 1끼리 GCD를 계산하고, gcd 값을 저장한다. 그리고 for문을 돌려주면서, GCD함수 안에 (i = 2~Input_size) gcd와 i를 넣어주고 계산하여 array 안에 있는 모든 데이터들의 gcd 값을 계산하여 줍니다.

```
int findGCD(int x, int y) {
  while(1) {
    int r = x%y;
    if(r == 0) return y;
    x = y;
    y = r;
  }
}
```

```
int GCD = findGCD(arr[1], arr[0]);
for(int i=2; i<size; i++) {
  GCD = findGCD(arr[i], GCD);
}
cout << "GCD of input is : " << GCD << '\n';
```

소수 개수 구하기

소수 개수를 구하는 법은 '에라토스테네스의 체'를 이용하면 쉽게 구할 수 있습니다.

해당 입력의 크기는 100,000 이하의 자연수만 받음으로, 100,000 크기의 배열을 만들고, 해당 수가 소수라면 해당 index의 값은 index에 넣어주고, 소수가 아니라면 0으로 값을 넣어줍니다.
또한 해당 수는 100,000의 `sqrt()` 함수를 씌워줘서 불필요한 데이터 access를 건너뛰게 해줍니다.

```
void primeNumber() { //find prime number and memorize into 'isPrime' vector
  for(int i=2; i<MAX; i++) isPrime[i] = i;
  for(int i=2; i<=sqrt(MAX); i++) {
    if(isPrime[i] == 0) continue;
    for(int j= i+i; j<MAX; j+=i) isPrime[j] = 0;
  }
}
```

실행 결과

입력 1 : 예제 입력시

> 11
> 368 12 58 74 712 12 38 1110 1612 4 222

입력 2 : 크기 작은 데이터 입력

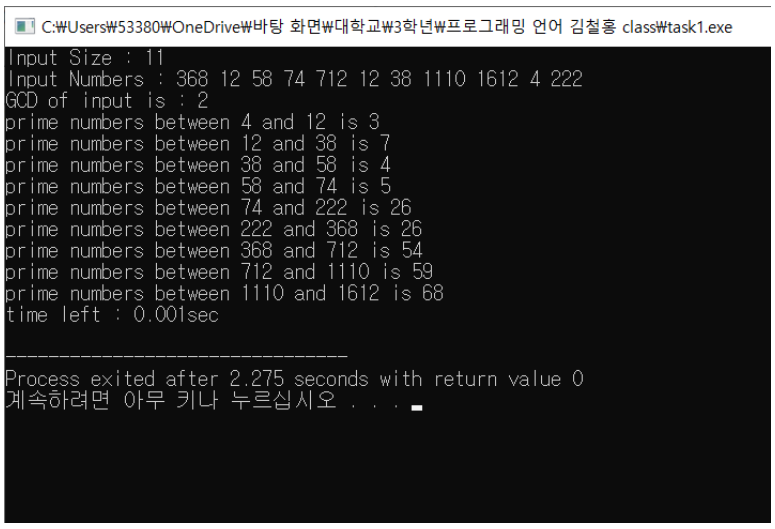
> 5
> 11 98 157 333 452

입력 3 : 임의의 거대한 크기의 데이터

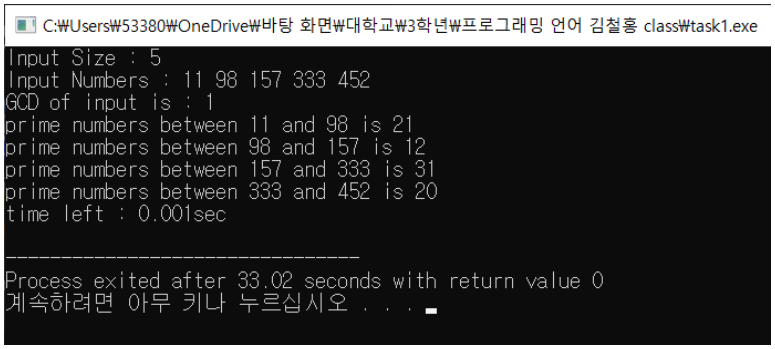
> 30
> 56171 72626 34830 91535 612 59433 87207 51536 95906
> 97516 16957 66157 31692 52866 70134 78850 22530 7410
> 57441 32985 33805 58375 23823 3097 17200 92860 96607
> 1975 65367 77938

C++

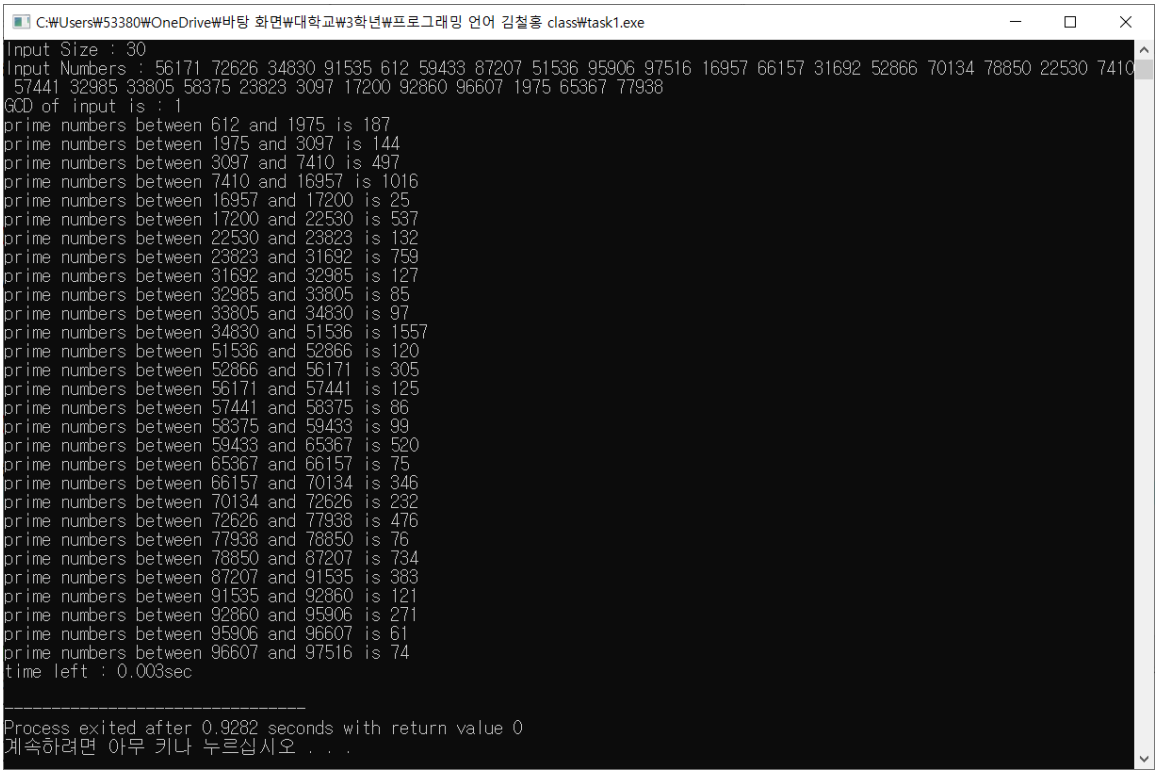
1. 예제 입력



2. 크기 작은 데이터 입력



3. 임의의 거대한 크기의 데이터



Java

1. 예제 입력

```
Input Size :
11
Input numbers processed :
368 12 58 74 712 12 38 1110 1612 4 222
GCD of input numbers is 2
Numbers of prime numbers between 4, 12: 3
Numbers of prime numbers between 12, 38: 7
Numbers of prime numbers between 38, 58: 4
Numbers of prime numbers between 58, 74: 5
Numbers of prime numbers between 74, 222: 26
Numbers of prime numbers between 222, 368: 26
Numbers of prime numbers between 368, 712: 54
Numbers of prime numbers between 712, 1110: 59
Numbers of prime numbers between 1110, 1612: 68
Execution time : 0.003 sec
```

2. 크기 작은 데이터 입력

```
Input Size :
5
Input numbers processed :
11 98 157 333 452
GCD of input numbers is 1
Numbers of prime numbers between 11, 98: 21
Numbers of prime numbers between 98, 157: 12
Numbers of prime numbers between 157, 333: 31
Numbers of prime numbers between 333, 452: 20
Execution time : 0.003 sec
```

3. 임의의 거대한 크기의 데이터

```
Input Size :
30
Input numbers processed :
56171 72626 34830 91535 612 59433 87207 51536 95906 97516 16957 66157 31692 52866 70134 78850 22530 7410 57441 32985 33805 58375 23823 3097 17200 92860 96607 1
GCD of input numbers is 1
Numbers of prime numbers between 612, 1975: 187
Numbers of prime numbers between 1975, 3097: 144
Numbers of prime numbers between 3097, 7410: 497
Numbers of prime numbers between 7410, 16957: 1016
Numbers of prime numbers between 16957, 17200: 25
Numbers of prime numbers between 17200, 22530: 537
Numbers of prime numbers between 22530, 23823: 132
Numbers of prime numbers between 23823, 31692: 759
Numbers of prime numbers between 31692, 32985: 127
Numbers of prime numbers between 32985, 33805: 85
Numbers of prime numbers between 33805, 34830: 97
Numbers of prime numbers between 34830, 51536: 1557
Numbers of prime numbers between 51536, 52866: 120
Numbers of prime numbers between 52866, 56171: 305
Numbers of prime numbers between 56171, 57441: 125
Numbers of prime numbers between 57441, 58375: 86
Numbers of prime numbers between 58375, 59433: 99
Numbers of prime numbers between 59433, 65367: 520
Numbers of prime numbers between 65367, 66157: 75
Numbers of prime numbers between 66157, 70134: 346
Numbers of prime numbers between 70134, 72626: 232
Numbers of prime numbers between 72626, 77938: 476
Numbers of prime numbers between 77938, 78850: 76
Numbers of prime numbers between 78850, 87207: 734
Numbers of prime numbers between 87207, 91535: 383
Numbers of prime numbers between 91535, 92860: 121
Numbers of prime numbers between 92860, 95906: 271
Numbers of prime numbers between 95906, 96607: 61
Numbers of prime numbers between 96607, 97516: 74
Execution time : 0.012 sec
```

Python

1. 예제 입력

```
plz input your number
11
input numbers for calculation
368 12 58 74 712 12 38 1110 1612 4 222
gcd is  2
prime numbers of between  4  and  12  is  3
prime numbers of between  12  and  38  is  7
prime numbers of between  38  and  58  is  4
prime numbers of between  58  and  74  is  5
prime numbers of between  74  and  222  is  26
prime numbers of between  222  and  368  is  26
prime numbers of between  368  and  712  is  54
prime numbers of between  712  and  1110  is  59
prime numbers of between  1110  and  1612  is  68
time left :  0.017999649047851562 sec
```

2. 크기 작은 데이터 입력

```
plz input your number
5
input numbers for calculation
11 98 157 333 452
gcd is  1
prime numbers of between  11  and  98  is  21
prime numbers of between  98  and  157  is  12
prime numbers of between  157  and  333  is  31
prime numbers of between  333  and  452  is  20
time left :  0.01996755599975586 sec
```

3. 임의의 거대한 크기의 데이터

```
plz input your number
30
input numbers for calculation
56171 72626 34830 91535 612 59433 87207 51536 95906 97516 16957 66157 31692 52866 70134 78850 22530 7410 57441 32985 33805 58375 23823
3097 17200 92860 96607 1975 65367 77938
gcd is  1
prime numbers of between  612  and  1975  is  187
prime numbers of between  1975  and  3097  is  144
prime numbers of between  3097  and  7410  is  497
prime numbers of between  7410  and  16957  is  1016
prime numbers of between  16957  and  17200  is  25
prime numbers of between  17200  and  22530  is  537
prime numbers of between  22530  and  23823  is  132
prime numbers of between  23823  and  31692  is  759
prime numbers of between  31692  and  32985  is  127
prime numbers of between  32985  and  33805  is  85
prime numbers of between  33805  and  34830  is  97
prime numbers of between  34830  and  51536  is  1557
prime numbers of between  51536  and  52866  is  120
prime numbers of between  52866  and  56171  is  305
prime numbers of between  56171  and  57441  is  125
prime numbers of between  57441  and  58375  is  86
prime numbers of between  58375  and  59433  is  99
prime numbers of between  59433  and  65367  is  520
prime numbers of between  65367  and  66157  is  75
prime numbers of between  66157  and  70134  is  346
prime numbers of between  70134  and  72626  is  232
prime numbers of between  72626  and  77938  is  476
prime numbers of between  77938  and  78850  is  76
prime numbers of between  78850  and  87207  is  734
prime numbers of between  87207  and  91535  is  383
prime numbers of between  91535  and  92860  is  121
prime numbers of between  92860  and  95906  is  271
prime numbers of between  95906  and  96607  is  61
prime numbers of between  96607  and  97516  is  74
time left :  0.03151750564575195 sec
```

실행 시간 (요약)

C++

1. 예제	0.001 sec
2. 작은 크기	0.001 sec
3. 큰 데이터	0.003 sec

Java

1. 예제	0.002 sec
2. 작은 크기	0.003 sec
3. 큰 데이터	0.012 sec

Python

1. 예제	0.017 sec
2. 작은 크기	0.019 sec
3. 큰 데이터	0.031 sec

느낀 점

각 Readability, Writability, Reliability, Cost를 3가지 언어에 순서를 매겨서 생각해보았습니다.

Readability

Readability는 각각 언어들의 특성을 생각하면 1위 Python, 공동 2위 C++, Java였던것 같습니다.

Python 자체가 코드 자체가 짧고, 개인적인 생각으로 다른 언어들에 비해 자연어에 가까운 구현된 언어다 보니, 프로그래머가 이해하기도 쉽고, 이것을 유지보수하는 데에도 큰 영향을 줄 것 같았습니다.

C++와 Java는 둘다 객체지향적인 언어기도 하고, 각 프로그래머가 주력으로 하는 언어가 무엇인지에 따라서 코드를 이해하는 수준이 차이가 날 것으로 생각되었습니다. 그래서 같은 순위로 생각해보았습니다.

Writability

Writability는 1위 Python, 공동 2위 C++, Java로 생각하였습니다.

먼저, 작성 자체도 python 자체가 용이하였습니다. 각 언어들을 어떤 방식으로 작성하는가를 보았을 때, 위의 설명과 마찬가지로 자연어에 가까운 코드를 작성하다보니 작성 용이도를 비교했을 경우 가장 높았습니다.

C++와 Java의 경우 구현 자체가 각 언어들의 특성에 따라서 달라지기 때문에 동일한 순위로 생각하였습니다.

Reliability

Reliability는 1위 Java, 2위 C++, 3위 Python으로 생각하였습니다.

Java의 변수들의 자동형변환에 대해 가장 보수적으로 넣어주는 것을 보았을 때, 3가지 중에 사용자의 의도대로 잘 작성하는 것은 Java라고 생각하였습니다.

C++는 데이터들의 연산도 적절한 형변환으로 이루어지며, 적절하게 기계어 친화적인 코드와 3가지 언어들 중에서 가장 low level language로 생각하여, 2위로 생각하였습니다.

다음은 Python인데, 자연어에 가까운 언어로 작성하는 것이 readability나 writability에서는 더욱 좋지만, 이것이 reliability에서는 오히려 의도대로 작동되지 않을 확률이 높다고 생각하였습니다.

Cost

시간적인 비용만 놓고 본다면 C++ < Java < Python 순으로 가장 많이 들었습니다.

아무래도 C++가 3가지 언어들과 비교해보면 가장 기계어에 가까운 언어이다 보니, 수행시간이 다른 언어에 비해 절대적으로 빠른 것을 확인할 수 있었습니다.

하지만 언어를 분석하는 cost는 컴파일, 실행하는 시간 뿐만 아니라 프로그래머를 길러내는 시간과 비용 같은 것도 포함된다고 배웠습니다. 이를 포함하여 생각하면, 파이썬은 프로그래밍에 초보자들도 다른 언어에 비해 손쉽게 입문할 수 있는 언어라는 점을 감안하면, 전체적인 cost는 C++ < Python < Java로 생각하였습니다.