



Object-Oriented Programming Report

Assignment 1-2

Professor	Donggyu Sim
Department	Computer engineering
Student ID	2022202093
Name	Wonju moon
Class (Design / Laboratory)	1 / B (미수강시 0로 표기)
Submission Date	2023. 3. 31

서식 있음: 간격 없음, 눈금에 맞춤

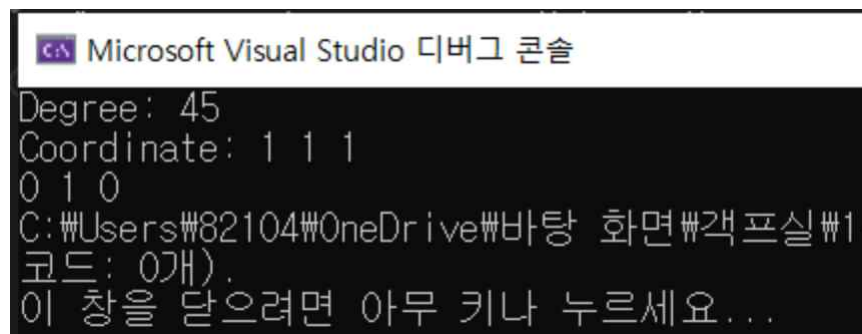
서식 있음: 글꼴: (영어)한컴 바탕,
(한글)한컴 바탕, 글꼴 색: 검정

Program 1

□ 문제 설명

주어진 행렬을 이용하여 만들어진 행렬 곱 T에 대해서 1X3행렬을 입력받아 행렬곱을 이행한다.

□ 결과 화면



```
Microsoft Visual Studio 디버그 콘솔
Degree: 45
Coordinate: 1 1 1
0 1 0
C:\Users\82104\OneDrive\바탕 화면\객프실\1
코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

cmath헤더파일 안 삼각함수의 역함수를 이용하여 PI를 정의한 후, 호도법을 통해 사용자로부터 입력받은 상수를 라디안 값으로 바꿔준다. 그 후 1차원 배열을 선언하여 사용자로부터 입력받은 Coordinate 값으로 정의해준다. 모든 행렬을 완성했으니 이중 반복문을 통해서 행렬의 곱연산 값을 출력해준다.

□ 고찰

처음에 T1 T2 T3값을 모두 입력 받아서 T를 계산한 후 새로운 행렬을 입력받아 다시 곱을 하는 문제인 줄 알았으나, 조교님께 문의한 결과 T1 T2 T3는 고정되어 있으므로 T를 따로 구하는 식은 작성하지 않아도 된다고 하셨습니다. 그래서 T1 T2 T3를 선언하지 않고 T를 바로 정의해서 문제 자체는 매우 쉬웠음. 출력할 때 int형 형변환을 통해서 double형을 int형으로 바꿔줬고 PI값을 정의할 때 살짝 고민이었는데 cmath헤더파일에 있는 삼각함수 역함수를 통해서 PI값을 정의해줬음.

Program 2

□ 문제 설명

병렬 저항과 직렬 저항을 구해서 옴의 법칙을 사용하여 전압 전류 파워 저항 등을 구하라.

□ 결과 화면

Microsoft Visual Studio 디버그 콘솔

```
Vs: 40
R1: 4
R2: 2
R(Load): 0
Vout: 80/6=13.3333
C:\Users\82104\Desktop\my code\Mentoring-00P\Wonju
드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

Microsoft Visual Studio 디버그 콘솔

```
Vs: 10
R1: 4
R2: 2
R(Load): 2
Vout: 10/5=2
Load_power_ratio: 10%
```

구현하지는 못했지만 순환소수를 구현하는 과정에서 나머지 값을 이용할 생각이었다. 순환소수는 애초에 값을 나눌 때 나머지 값이 같은 값이 나오면 순환하는 무한소수가 되기 때문에 배열에 값을 하나하나 담고 값을 나눌 때마다 배열에 담아둔 값과 비교해서 해당 같은 값이 있다면 해당 배열의 길이만큼 순환마디의 길이가 되지 않을까 하는 생각을 했다. 그러나 구현하지 못했다.

□ 고찰

구현하지는 못했지만 순환소수를 구현하는 과정에서 나머지 값을 이용할 생각이었다. 순환소수는 애초에 값을 나눌 때 나머지 값이 같은 값이 나오면 순환하는 무한소수가 되기 때문에 배열에 값을 하나하나 담고 값을 나눌 때마다 배열에 담아둔 값과 비교해서 해당 같은 값이 있다면 해당 배열의 길이만큼 순환마디의 길이가 되지 않을까 하는 생각을 했다. 구현하지 못한 아쉬움이 있다.

Program 3

□ 문제 설명

숫자를 사용자로부터 입력받고 해당 숫자가 2진법, 8진법, 10진법, 16진법 중에서 어떤 진법으로 쓰였는지 입력받는다. 그 후 해당 숫자를 다른 진법으로 변환하는 프로그램을 만든다. 기본적으로 일상생활에서 사용하는 10진법이 숫자 계산이 편하기 때문에 입력받은 숫자를 10진법으로 변환하는 함수와 해당 숫자를 다시 다른 진법으로 변환하는 함수를 짤다. 이때, 16진법은 문자가 들어가므로 예외로 두고 짤다.

□ 결과 화면

```
Microsoft Visual Studio 디버그 콘솔
181 10 2
10110101
C:\Users\82104\OneDrive\바탕 화면\객프
코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

```
Microsoft Visual Studio 디버그 콘솔
265 8 16
B5
C:\Users\82104\OneDrive\바탕 화면\객프
코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

```
Microsoft Visual Studio 디버그 콘솔
10110101 2 8
265
C:\Users\82104\OneDrive\바탕 화면\객프실
코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

Char형식의 배열을 이용하여 사용자로 부터 받은 입력값을 배열 인덱스 하나하나에 따로 저장한다. 그리고 아스키 코드를 이용하여 int형 배열에 char배열에서 입력받았던 값을 숫자로 저장한다. 이때 입력받은 숫자 중에서 16진수가 있다면 문자를 숫자로 바꾸는 것도 고려한다. 이때 change1함수에서 입력받은 숫자를 모두 10진수로 바꿔서 더해주고 change2에서 해당 값을 다시 사용자가 원하는 진법으로 바꿔준다. 마찬가지로 이때 16진수의 경우 문자는 int형으로 출력하기 어렵기 때문에 따로 고려해준다. 출력 과정의 경우 조건문과 재귀함수를 이용하여 작성한다.

□ 고찰

16진수를 출력하는 과정에서 왜인지는 모르겠지만 a 를 n^2 로 나눈 나머지가 잘
출력이 안 됐음. 새로운 변수를 만들어 `int temp = a%2`라고 설정해주면 안되지만
`a%=2`라고 설정하면 됐음. 이러한 차이가 왜 발생했는지 이해가 안됨.

Program 4

□ 문제 설명

Merge sort, bubble sort, insertion sort, quick sort를 구현하라. 그리고 그들간의 시간복잡도를 계산하
라.

□ 결과 화면

```
Microsoft Visual Studio 디버그 콘솔
15
26 9 23 19 1 35 4 31 14 13 28 3 18 33 19

1. bubble sort 2. insertion sort 3. merge sort 4. quick sort : 1
sorted order: 1 3 4 9 13 14 18 19 19 23 26 28 31 33 35
median number: 19
time: 0.001697700000 sec
```

Microsoft Visual Studio 디버그 콘솔

```
15
26 9 23 19 1 35 4 31 14 13 28 3 18 33 19

1. bubble sort 2. insertion sort 3. merge sort 4. quick sort : 2
sorted order: 1 3 4 9 13 14 18 19 19 23 26 28 31 33 35
median number: 19
time: 0.001483400000 sec
```

Microsoft Visual Studio 디버그 콘솔

```
15
26 9 23 19 1 35 4 31 14 13 28 3 18 33 19

1. bubble sort 2. insertion sort 3. merge sort 4. quick sort : 3
sorted order: 1 3 4 9 13 14 18 19 19 23 26 28 31 33 35
median number: 19
time: 0.001646500000 sec
```

Microsoft Visual Studio 디버그 콘솔

```
15
26 9 23 19 1 35 4 31 14 13 28 3 18 33 19

1. bubble sort 2. insertion sort 3. merge sort 4. quick sort : 4
sorted order: 1 3 4 9 13 14 18 19 19 23 26 28 31 33 35
median number: 19
time: 0.001655100000 sec
```

999개 난수 비교

1. bubble sort

```
928 928 929 930 931 932 933 936 937 938 939 940 941 94
960 961 962 963 963 964 964 965 965 965 967 967 971 97
987 987 987 988 988 991 991 992 996 997 998 998 1000
median number: 494
time: 0.120363200000 sec
```

2. insertion sort


```
928 928 928 938 937 932 933 938 937 938 933 948 947 942 9
960 961 962 963 963 964 964 965 965 965 967 967 971 972 9
987 987 987 988 988 991 991 992 996 997 998 998 1000
median number: 494
time: 0.118483400000 sec
```

3. merge sort

```
928 928 928 938 937 932 933 938 937 938 933 948 947 942 973
960 961 962 963 963 964 964 965 965 965 967 967 971 972 973
987 987 987 988 988 991 991 992 996 997 998 998 1000
median number: 494
time: 0.119574200000 sec
```

4. quick sort

```
960 961 962 963 963 964 964 965 965 965 967 967 971 972
987 987 987 988 988 991 991 992 996 997 998 998 1000
median number: 494
time: 0.121076800000 sec
```

□ 고찰

1. 시간복잡도

주어진 예시에서는 merge sort가 가장 빠른 기록을 보여주지는 않았다. merge sort는 정렬할 배열의 길이가 길수록 훨씬 더 빠른 정렬을 보여주었다. 이유는 다른 sort는 기준을 잡고 하나하나 비교하고 swap하는 작업을 반복하는데 merge sort의 경우에는 일단 다 떨어뜨려 놓은 후에 비교를 하기 때문에 주어진 배열이 길다면 훨씬 빠른 속도로 정렬할 수 있다. 가장 기본적인 배열은 bubble sort인데 bubble sort의 경우의 시간 복잡도는 N^2 초다. 또한, insertion sort 또한 bubble sort와 크게 다르지 않게

하나하나 다른 인덱스의 배열과 비교하면서 교환하는 형태를 띄기 때문에 시간 복잡도는 N^2 초만큼 걸린다고 할 수 있다. 그러나 quick sort의 경우에는 피벗을 설정하고 피벗에 맞게 간단하게 정렬한 후 절반씩 빠르게 정렬하기 때문에 훨씬 시간복잡도가 덜하다. 물론 최악의 경우 모든 경우의 수를 비교해야 하기 때문에 최대 N^2 만큼의 시간이 걸리지만, 그럼에도 불구하고 insertion sort나 bubble sort보다는 빠르게 정리할 수 있다는 가능성을 보여준다. 이때 pivot을 적당한 값에 선택했다면 $N \log N$ 만큼의 시간 복잡도를 갖게 된다. 따라서 앞의 두 정렬보다는 훨씬 빠른 시간 복잡도를 보여준다. 그러나 pivot의 값을 잘못 선택했다면 N^2 만큼의 시간 복잡도를 보여준다는 점에서 merge sort보다는 성능이 좋지 않다. merge sort는 항상 $N \log N$ 만큼의 시간 복잡도를 보여주는데, quick sort는 최선의 선택일 때 $N \log N$ 이 걸린다고 한다면 merge sort는 $N \log N$ 의 시간복잡도를 보장하는 점에 있어서 quick sort보다 이점이 있다고 할 수 있다.

결국 평균적인 시간복잡도를 비교해본다면

$\text{bubble sort} \leq \text{insertion sort} < \text{quick sort} < \text{merge sort}$ 라고 볼 수 있는데, insertion sort의 경우 만약 최선의 경우가 선택된다면 N 이라는 말도 안되는 속도를 구현할 수 있게 된다.

2. 시간복잡도 외에 장점 및 단점

우선, bubble sort의 장점으로서는 구현이 쉽다는 점이 있다. 그러나 역시나 시간이 오래 걸리고 효율적이지 못하기 때문에 좋은 분류 방법은 아니라고 할 수 있다. insertion sort의 경우에는 최악의 경우 N^2 이라는 느린 시간복잡도를 갖는다는 단점이 있고, 데이터의 상태에 따라서 성능 차이가 심하다는 단점 또한 갖고있다. 그럼에도 불구하고 최선의 경우 N 이라는 매우 빠른 시간 복잡도를 갖고 있고, 성능이 좋기 때문에 다른 정렬법에 일부로 사용되기도 한다. quick sort의 경우에는 실행시간이 $N \log N$ 으로 빠른 편에 속한다는 장점이 있다. 그러나 pivot이 무엇이냐에 따라 성능 차이가 매우 심하고 최악의 경우 N^2 의 시간이 걸린다는 단점이 있다. 또한, merge sort의 경우에는 임시적인 배열을 하나 더 만들어야 하기 때문에 추가적인 메모리 공간이 필요하다는 단점이 있다. 물론 가장 큰 장점은 빠른 편의 시간복잡도이다.

merge sort와 quick sort를 구현하는 과정에서 재귀함수가 사용되는데, 재귀함수의 조건부가 상당히 길었다. 그렇기에 이해하는데 꽤 오랜 시간이 필요했다. 또한 merge

sort의 경우 서로 다른 두 개의 함수를 재귀함수로 호출하는 과정이 있어서 더욱 어려움을 느꼈다. merge_sort라는 함수에서 모든 배열을 잘게 쪼개주고 merge라는 함수에서 배열 인자를 비교하고 정렬해서 붙여주는 식으로 구현했다. 기본적인 아이디어는 중간 값을 기준으로 좌우를 나누어 서로 다른 배열의 값을 꾸준히 비교하면서 오름차순으로 임시 배열에 저장해준 후에 배열을 한번에 복제하는 정렬 방식이었다. 세 가지 변수를 선언하여 i는 시작점 j는 중앙 다음 값 k는 임시 배열의 시작점으로 선언하여 sort[i]와 sort[j]를 끊임없이 비교해준다. 이때 하나씩 sorted[k]값에 오름차순으로 담고 i와 j의 인덱스 값을 하나씩 증가시켜준다. i는 mid까지 j는 중간 다음 값까지 간다. 또한, i든 j든 sorted[k]에 값이 삽입되면 k또한 값을 하나 증가시켜준다. 그래야 다음 값을 비교하여 차례대로 쌓을 수 있기 때문이다. 이때, i가 먼저 mid에 도달할 수도 있고 j가 먼저 n에 도달할 수 있다. 이러한 경우 비교할 값이 사라지기 때문에 남아있는 쪽의 배열을 sorted[k]에 그대로 복제해주면 된다. 재귀함수는 이미 결과가 정해져있다고 가정하고 시작하기 때문에 한 쪽의 경우는 이미 정렬이 완료되어 있다. 따라서, 그대로 복제해준다면 정렬은 끝난 것이다. 그 후 마지막에 원배열이었던 sort배열에 sorted를 그대로 카피해준다.

문제에서 주어진 값들은 시간복잡도를 구하기 너무 어려웠다. 우선 배열의 길이가 너무 짧아서 초단위로 구현되지가 않았다. 999개까지 배열을 늘려봤지만 이론값과 실험결과값은 다르게 나왔다. 실제로는 insertion sort가 가장 빠르게 나왔는데 아마 최선의 경우에 가까운 경우가 발생한 것이 아닐까 하는 생각이 든다. 의아한 점은 bubble sort의 경우 무조건 N^2 의 시간이 걸리므로 가장 느려야 하지만 실제로는 가장 느린 결과값이 나오지 않았다는 것이다.