

REPORT

순천향대학교



과목명 1 이산수학
담당교수 1 박득순 교수님
학과 1 컴퓨터소프트웨어 공학과
학년 1 2학년
학번 1 20164006
이름 1 박 혼
제출일 1 3월 23일 목요일

2017년 3월 9일 교수님께서 마술 카드(magic card)를 보여주었다. 주로 내가 아는 마술에서는 많은 과학적 트릭이나 사람들의 눈속임을 위한 마술사들의 많은 노력과 연습을 통해서 가능해진 마술들이었다. 하여 어떠한 트릭을 사용하는지 교수님의 손을 눈을 부릅뜨고 관찰을 하였다. 그러나 결코 손의 수상한 움직임이나 트릭 등을 절 때 찾을 수가 없었다. 카드를 아무리 눈을 뜨고 보아도 카드의 수상한 점도 발견을 할 수가 없었고, 교수님께서서는 심지어 재 옆에 앉은 친구들에게 카드를 섞을 수 있는 기회를 주기도 하였다. 어떻게 이러한 일이 가능할 수가 있었을까? 또한 교수님께서서는 c언어를 사용하여 작성한 프로그램으로도 마술을 보여주셨다. 프로그램으로 코딩을 하기 위해서는 그에 대한 수학적인 알고리즘이 요구된다. 나는 내가 생각한 수가 5장의 카드에 있는 지만을 확인한 것뿐인데 어떻게 알고리즘을 구현하면 컴퓨터가 내가 생각하는 수를 맞출 수 있을까? 전 시간에 교수님께서 수학은 현상을 추상화하여 수식을 만들어 해결하고 정의하는 학문인 수학이라 알려주셨다. 한마디로 마술 카드는 수학의 수식을 통하여 정의가 가능하다는 것을 의미한다. 이번 레포트를 통해 재가 연구하고 분석했던 모든 것을 기록해 보겠다.

목차

I. 서론	5
1. 마술 카드란?	5
1-1. 마술 카드의 방식	5
1-2. 3월 9일의 마술 카드를 재현해 보자	5
2. 프로젝트 목표	5
3. 프로그램 구상	5
4. 교수님의 조언	6
II. 본론	7
1. 마술 카드 로직 찾기	7
1-1. 마술 카드의 숫자 맞추기	7
1-2. 마술 카드 안의 숫자	8
2. 정적 배열을 사용한 문제해결 방식(1 단계)	10
2-1. 문제해결 방식 (1차)	10
2-1-1. 마술 카드 안의 숫자의 규칙	10
2-1-2. 소스 코드	11
2-1-3. 디버깅 결과	12
2-1-4. 차트를 통한 분석	12
2-2. 문제해결 방식 (2차 – 비트 연산의 곱셈)	13
2-2-1. 비트 연산자의 곱셈 방식	13

2-2-2. 소스 코드	13
2-2-3. 디버깅 결과.....	15
2-2-4. 차트를 통한 분석.....	15
2-3. 문제해결 방식 (3차 – 재귀 호출)	16
2-3-1. 재귀 호출이란?.....	16
2-3-2. 소스 코드	17
2-3-3. 디버깅 결과.....	18
2-3-4. 재귀 호출의 오버플로우의 발생.....	18
2-4. 문제해결 방식 (4차 – 오버플로우 극복 알고리즘).....	19
2-4-1. 오버플로우 발생 && 개선 방안.....	19
2-4-2. 소스 코드	21
2-4-3. 디버깅 결과.....	22
2-4-4. 개선 결과 분석.....	22

Ⅲ. 결론	23
1. 1 단계 최종 결과.....	23
1-1. 최종 소스.....	23
1-2. 1차 레포트를 마치며 느낀 점	24

I. 서론

1. 마술 카드란?

페이지 | 5

1-1. 마술 카드의 방식

n장의 카드에서 0~2의 n승 -1 숫자들 중에 하나를 고른 수를 카드를 통해서 맞추는 마술입니다.

1-2. 3월 9일의 마술 카드를 재현해 보자

카드로 직접 하는 마술이 잘 보이지가 않아서 c언어로 디자인까지 훌륭한 프로그램에 몰입하여 보았다. 우선 n장의 카드를 입력을 한 후에 나는 숫자 하나를 골랐다. 그 후 A~E까지의 카드에서 내가 고른 숫자가 있는지 없는지 대답을 하였다. 5장의 카드를 선택하였기에 나올 수 있는 경우의 수는 32가지이다. 그 중에 내가 고른 숫자는 1가지이고 다시 말해 컴퓨터가 내가 고른 숫자를 맞출 확률은 32:1이었던 것이다. 이러한 확률에 5장의 카드를 사용하여 맞추니 신기할 따름 이었다.

2. 프로젝트 목표

1단계 - 기본적인 n장의 마술 카드 구현(교재의 5장 카드 프로그래밍 참조)하고(정적 할당)

반드시 **overflow 발생시키고**, overflow를 처리하는 방법, 얼마까지 가능한지?

실행 시간과 기억 장소 확인

2단계 - 동적 할당으로 바꾸어 보기? 그럴 때의 장단점은?

3단계 - 카드를 실제 카드와 비슷하게 만들어 보기

3. 프로그램 구상

1. n장의 카드를 입력을 받는다.
2. 사용자에게 고를 수 있는 수의 범위를 알려주고 숫자를 고르도록 하는 문구를 출력한다.
3. n장의 카드를 보여준다.
4. 사용자가 고른 카드가 있는지의 여부를 받는다.
5. 사용자가 고른 숫자를 맞춘다.

4. 교수님의 조언

3 가지를 했습니다.

리포트에서는 좀더 내용을 잘표현하기 위해서 문제분석에서 뭐를 할 것인지 작성하세요.
그리고 3 가지를 해봤는데 그것들에 대한 결과 화면들이 없고, 이론적으로도 분석되지 않았읍니다.

각각을 수행했을 때 화면 결과와 실행 시간 메모리 체크하고 그래프로 그려보고, 이론적으로 따져보세요.

overflow 로 해결했다고 했는데 무엇을 어떻게 해결했는지 하나도 내용이 없습니다. 그렇게 해결했을 때 얼마정도 더 할 수 있었는지 왜 그렇게 밖에 안되는지도 확인하세요

II. 본론

1. 마술 카드 로직 찾기

페이지 | 7

1-1 카드의 숫자 맞추기 방식

몇가지 경우를 통해서 로직을 찾아보려고 한다.

#1 Case - 5장의 카드에서 내가 고른 수가 7이면 대답을 어떻게 하게 될까?

A카드 - 예

B카드 - 예

C카드 - 예

D카드 - 아니요

E카드 - 아니요

#2 Case - 5장의 카드에서 내가 고른 수가 12이면 대답을 어떻게 하게 될까?

A카드 - 아니요

B카드 - 아니요

C카드 - 예

D카드 - 예

E카드 - 아니요

#3 Case - 5장의 카드에서 내가 고른 수가 31이면 대답을 어떻게 하게 될까?

A카드 - 예

B카드 - 예

C카드 - 예

D카드 - 예

E카드 - 예

표로 나타낸 결과이다.

	E	D	C	B	A
#1 case	no	no	yes	yes	yes
#2 case	no	yes	yes	no	no
#3 case	yes	yes	yes	yes	yes

내가 고른 수를 이진수로 표현해 보자

$$7 = 111$$

$$12 = 1100$$

$$31 = 11111$$

표에 표현된 yes를 1로 no를 0으로 바꾸어 생각한다면 이진수로 만든 수와 똑같다는 점을 볼 수가 있었다.

1-2 마술 카드 안의 숫자

카드 안의 숫자 어떠한 의미가 있는 숫자일까?

5장의 카드를 뽑는다 가정을 하겠다.

A카드 - 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31 (숫자 16개)

B카드 - 2, 3, 6, 7, 10, 11, 14, 15, 18, 19, 22, 23, 26, 27, 30, 31 (숫자 16개)

C카드 - 4, 5, 6, 7, 12, 13, 14, 15, 20, 21, 22, 23, 28, 29, 30, 31 (숫자 16개)

D카드 - 8, 9, 10, 11, 12, 13, 14, 15, 24, 25, 26, 27, 28, 29, 30, 31 (숫자 16개)

E카드 - 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31 (숫자 16개)

그냥 보기에는 아무 의미 없는 숫자처럼 보인다.

카드의 숫자의 비밀을 파헤쳐 보기 위해 숫자가 출력된 부분을 1 출력이 안된 부분을 0으로 표를 작성해 보았다

.

0~31까지의 숫자들과 카드의 관계를 표로 만들어 보았다.

	E	D	C	B	A
0	0	0	0	0	0
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	0	1	1
4	0	0	1	0	0
5	0	0	1	0	1
6	0	0	1	1	0
7	0	0	1	1	1
8	0	1	0	0	0
9	0	1	0	0	1
10	0	1	0	1	0
11	0	1	0	1	1
12	0	1	1	0	0
13	0	1	1	0	1
14	0	1	1	1	0
15	0	1	1	1	1
16	1	0	0	0	0
17	1	0	0	0	1
18	1	0	0	1	0
19	1	0	0	1	1
20	1	0	1	0	0
21	1	0	1	0	1
22	1	0	1	1	0
23	1	0	1	1	1
24	1	1	0	0	0
25	1	1	0	0	1
26	1	1	0	1	0
27	1	1	0	1	1
28	1	1	1	0	0
29	1	1	1	0	1
30	1	1	1	1	0
31	1	1	1	1	1

표를 통해 출력되는 기준이 이진수의 자릿수와 관계가 있다는 점을 한눈에 볼 수가 있었다,

2. 정적 배열을 사용한 문제해결 방식

2-1 문제해결 방식(1차)

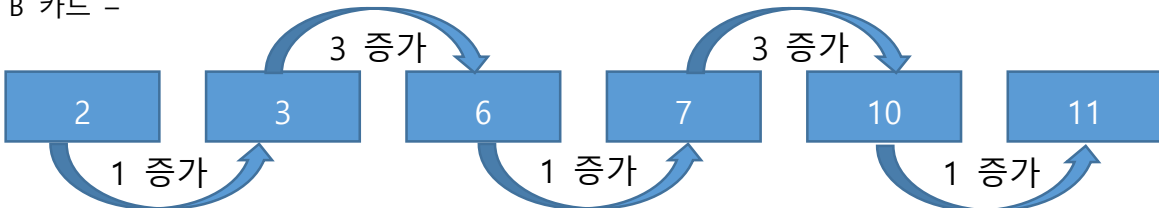
2-1-1 마술 카드 안의 숫자의 규칙

카드 속의 숫자들 증가 값을 비교해 보겠습니다.

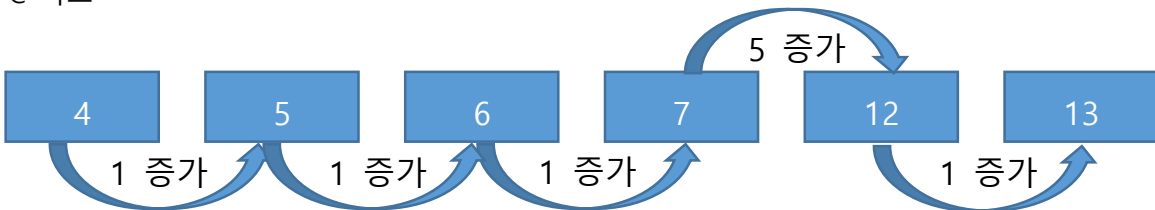
A 카드 -



B 카드 -



C 카드 -



그림의 정리를 통해 하나의 규칙을 찾을 수가 있었습니다.

첫번째 A카드를 분석을 보자 A의 0번지에 들어있는 값은 1이다. 1증가가 몇 번 시도가 되었나 0번 시도가 되었습니다. 그 후 숫자는 몇이 증가 되었나? 2가 증가했습니다.

다음은 두번째 카드 B를 분석을 해보자 B의 0번지에 들어있는 값은 2이다. 1증가가 몇 번 시도가 되었나 1번 시도가 되었습니다. 그 후 숫자는 몇이 증가 되었나? 3이 증가했습니다.

마지막으로 세번째 카드 C를 분석을 해보자 C의 0번지에 들어있는 값은 4이다. 1증가가 몇 번 시도가 되었나 3번 시도가 되었습니다. 그 후 숫자는 몇이 증가 되었나? 5가 증가했습니다.

이와 같은 결론으로 1씩 증가 횟수 = 0번지 수 -1번, 그 후 증가되는 숫자의 값은 = 0번지 수 +1 의 공식을 세우게 되었습니다.

2-1-2 소스 코드

```
1  /*
2  20164006-박훈-마술카드(정적배열)-1차
3  */
4  #include<stdio.h>
5  #include<math.h>
6  int main(void)
7  {
8      int magic_card[239000] = { 0 };
9      int ij;
10     int n;
11     int cnt=1,sum=0;
12     int x;
13     printf("몇장의 카드를 사용하시겠습니까?\n");
14     scanf("%d", &n);
15     printf("0~%d카드중에 숫자를 하나 고르세요(고른 후 엔터를 누르세요)\n",(int)pow(2,n)-1);
16     for (i = 0; i < n; i++)
17     {
18         printf("%c카드 - 고른 숫자가 있으면 1 아니면 0을 입력하시오\n",'A'+i);
19         magic_card[0] = (int)pow(2, i);
20         printf("%d ", magic_card[0]);
21         cnt = 1;
22         for (j = 1; j < (int)pow(2, n - 1); j++)
23         {
24             if (cnt == magic_card[0])
25             {
26                 magic_card[j] = magic_card[j - 1] + cnt + 1;
27                 cnt = 1;
28             }
29             else
30             {
31                 magic_card[j] = magic_card[j - 1] + 1;
32                 cnt++;
33             }
34             printf("%d ", magic_card[j]);
35         }
36         printf("\n고른 숫자가 있으면 1 아니면 0을 입력해 주세요>>");
37         scanf("%d", &x);
38         if(x==1)
39             sum = sum + (int)pow(2, i);
40     }
41     printf("당신이 고른 숫자는 : %d 입니다.", sum);
```

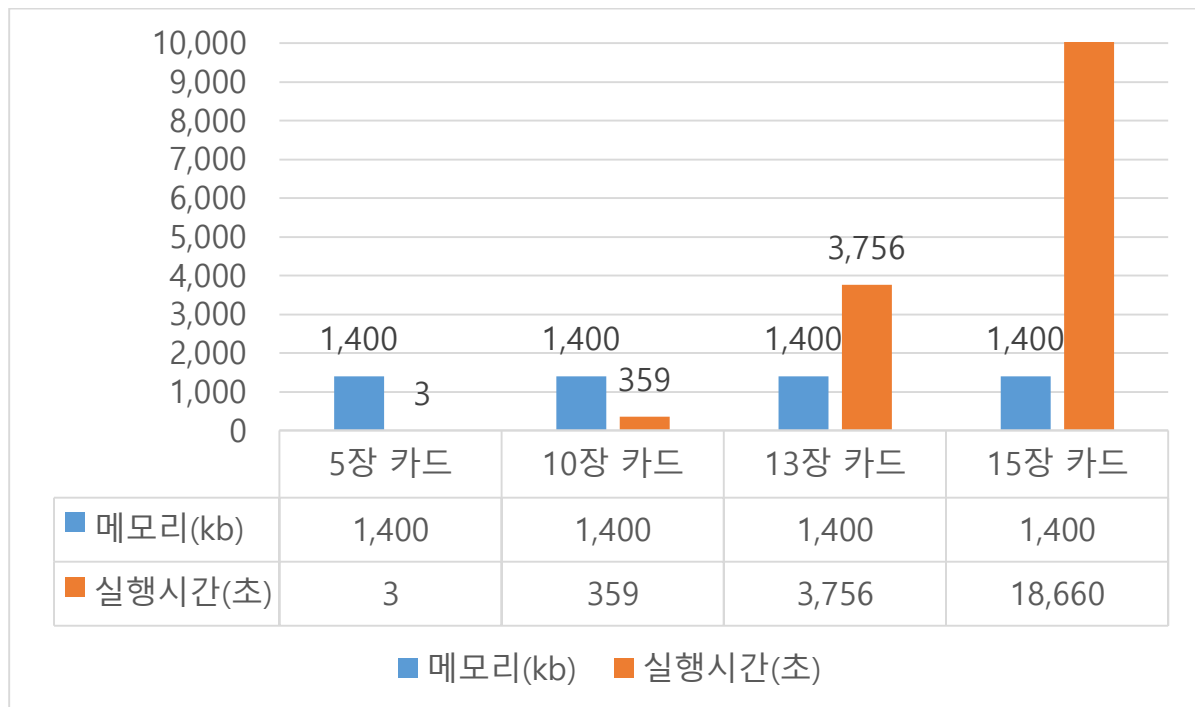
```
42     return 0;
43 }
```

2-1-3 디버깅 결과

페이지 | 12

```
C:\WINDOWS\system32\cmd.exe
몇장의 카드를 사용하시겠습니까?
5
0~31카드중에 숫자를 하나 고르세요(고른 후 엔터를 누르세요)
A카드 - 고른 숫자가 있으면 1 아니면 0을 입력하십시오
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31
고른 숫자가 있으면 1 아니면 0을 입력해 주세요>>1
B카드 - 고른 숫자가 있으면 1 아니면 0을 입력하십시오
2 3 6 7 10 11 14 15 18 19 22 23 26 27 30 31
고른 숫자가 있으면 1 아니면 0을 입력해 주세요>>1
C카드 - 고른 숫자가 있으면 1 아니면 0을 입력하십시오
4 5 6 7 12 13 14 15 20 21 22 23 28 29 30 31
고른 숫자가 있으면 1 아니면 0을 입력해 주세요>>1
D카드 - 고른 숫자가 있으면 1 아니면 0을 입력하십시오
8 9 10 11 12 13 14 15 24 25 26 27 28 29 30 31
고른 숫자가 있으면 1 아니면 0을 입력해 주세요>>0
E카드 - 고른 숫자가 있으면 1 아니면 0을 입력하십시오
16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
고른 숫자가 있으면 1 아니면 0을 입력해 주세요>>0
당신이 고른 숫자는 : 7 입니다.계속하려면 아무 키나 누르십시오 . . .
```

2-1-4 차트를 통한 분석



차트를 통해 카드의 장수와 실행속도가 비례함을 볼 수가 있었다. 또한 장수가 많아져도 똑같이 1.4mb(1,400 kb)의 메모리가 스택에 계속 사용됨을 볼 수가 있었다.

2-2 문제해결 방식(2차 – 비트 연산의 곱셈)

2-2-1 비트 연산의 곱셈 방식

1차 문제해결 방안에서는 $2*2*2$ 이런 방식의 문제를 c언어 라이브러리에 있는 `math.h` 헤더 파일의 `pow` 함수를 사용하여 지수 계산을 시도하였다.

이번에는 이 방식을 비트 연산의 곱셈 방식으로 바꿔보려고 한다.

방법은 아래의 표와 같다.

	지수 계산	비트 계산
2	2	000010
$2*2$	$2*2$	000100
$2*2*2$	$2*2*2$	001000
$2*2*2*2$	$2*2*2*2$	010000
$2*2*2*2*2$	$2*2*2*2*2$	100000

다음 표를 보면 비트의 자릿수를 한자리씩 이동하는 것과 지수 계산의 결과가 같음을 볼 수가 있다.

이번 방식은 위와 같은 비트 연산을 하기 위해 시프트 연산자인 '<<'를 사용해볼 것이다.

2-2-2 소스 코드

```
1  /*
2  20164006-박훈-마술카드(정적배열)-2차(비트연산)
3  */
4  #include<stdio.h>
5  int main(void)
6  {
7      double duration;
8      int magic_card[239000] = { 0 };
9      int i, j;
10     int n;
11     int cnt = 1, sum = 0;
12     int x = 0;
13     printf("몇장의 카드를 사용하시겠습니까?\n");
14     scanf("%d", &n);
15     printf("0~%d카드중에 숫자를 하나 고르세요(고른 후 엔터를 누르세요)\n", (1<n) - 1);
16     for (i = 0; i < n; i++)
17     {
18         printf("%c카드 - 고른 숫자가 있으면 1 아니면 0을 입력하시오\n", 'A' + i);
19         magic_card[0] = 1<i;
20         printf("%d ", magic_card[0]);
21         cnt = 1;
22         for (j = 1; j < 1<n-1; j++)
23         {
24             if (cnt == magic_card[0])
25             {
26                 magic_card[j] = magic_card[j - 1] + cnt + 1;
27                 cnt = 1;
28             }
29             else
30             {
31                 magic_card[j] = magic_card[j - 1] + 1;
32                 cnt++;
33             }
34             printf("%d ", magic_card[j]);
35         }
36         printf("\n고른 숫자가 있으면 1 아니면 0을 입력해 주세요>>");
37         scanf("%d", &x);
38         if (x == 1)
39             sum = sum + (1<i);
40     }
41     printf("당신이 고른 숫자는 : %d 입니다.", sum);
```

```
42     return 0;
43 }
```

2-2-3 디버깅 결과

C:\WINDOWS\system32\cmd.exe

몇장의 카드를 사용하시겠습니까?

5

0~31카드중에 숫자를 하나 고르세요(고른 후 엔터를 누르세요)

A카드 - 고른 숫자가 있으면 1 아니면 0을 입력하십시오

1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31

고른 숫자가 있으면 1 아니면 0을 입력해 주세요>>1

B카드 - 고른 숫자가 있으면 1 아니면 0을 입력하십시오

2 3 6 7 10 11 14 15 18 19 22 23 26 27 30 31

고른 숫자가 있으면 1 아니면 0을 입력해 주세요>>1

C카드 - 고른 숫자가 있으면 1 아니면 0을 입력하십시오

4 5 6 7 12 13 14 15 20 21 22 23 28 29 30 31

고른 숫자가 있으면 1 아니면 0을 입력해 주세요>>1

D카드 - 고른 숫자가 있으면 1 아니면 0을 입력하십시오

8 9 10 11 12 13 14 15 24 25 26 27 28 29 30 31

고른 숫자가 있으면 1 아니면 0을 입력해 주세요>>1

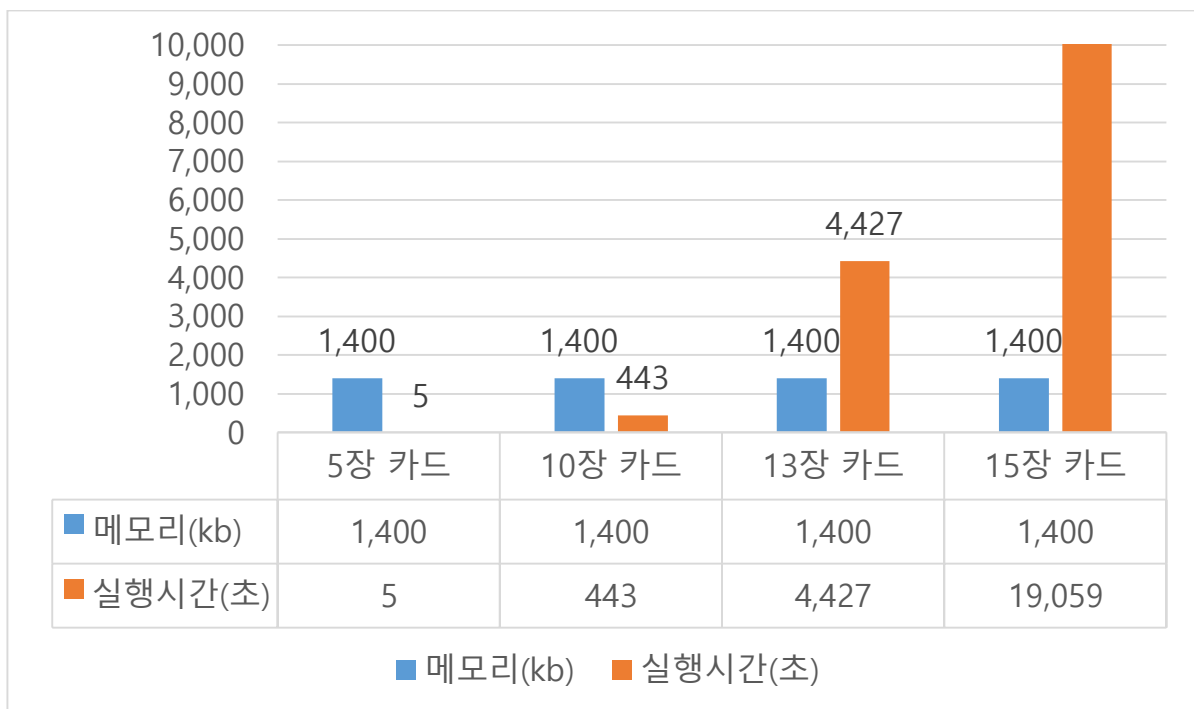
E카드 - 고른 숫자가 있으면 1 아니면 0을 입력하십시오

16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

고른 숫자가 있으면 1 아니면 0을 입력해 주세요>>1

당신이 고른 숫자는 : 31 입니다.계속하려면 아무 키나 누르십시오 . . .

2-2-4 차트를 통한 분석



내 생각과 결과가 다르게 나와서 많이 놀라웠다. 당연히 제곱연산보다 빠를 것이라 생각을 하고 코딩을 하였지만 1차 차트와 비교하였을 때 비트 연산이 더 느린 것으로 확인이 되었다. 이것의 결과를 보니 프로그램은 생각만 할 것이 아니라 직접 소스를 짜보고 확인해야 한다는 생각이 든다.

2-3 문제해결 방식(3차 – 재귀 호출)

2-3-1 재귀 호출이란?

재귀 함수는 함수 내에서 자기자신을 다시 호출하는 함수를 의미한다.

2학년 현재 듣는 많은 과목에서 재귀 함수에 대한 설명을 많이 들을 수가 있었다.

많은 교수님들께서 재귀 함수는 속도가 느리고 많은 메모리의 사용으로 함수 호출에 대한 오버헤드가 발생 시킬 수 있다는 설명을 들어왔다.

하여 책에 있는 반복문을 재귀 호출로 코딩을 하는 연습을 해보았지만 일반적으로 재귀 함수를 사용할 때랑 반복문을 사용할 때의 차이점을 크게 느끼지 못하였다.

마술 카드의 이정도의 정적 배열이면 차이를 느낄 수 있지 않을까 하는 마음에 반복문을 재귀 함수로 바꾸어 반복문으로 코딩을 하였을 때와 재귀 함수로 구현을 했을 때의 차이를 직접 느껴보고자 하는 마음으로 도전을 해보았다.

2-3-2 소스 코드

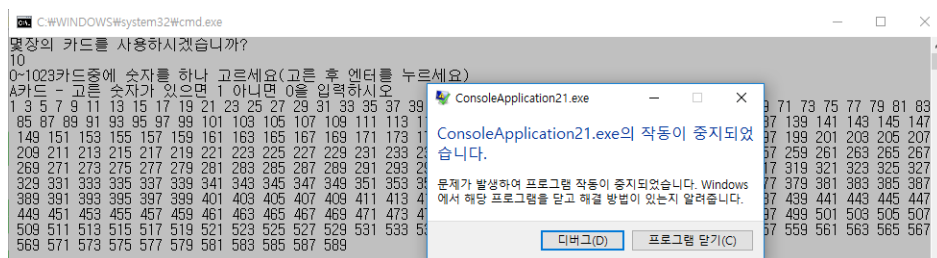
```
1  #include<stdio.h>
2  #include<math.h>
3  void print(int n, int i,int magic_card[],int cnt)
4  {
5      if (i==(int)pow(2, n - 1))
6          return 0;
7      if (cnt == magic_card[0])
8      {
9          magic_card[i] = magic_card[i - 1] + cnt + 1;
10         cnt = 0;
11     }
12     else
13         magic_card[i] = magic_card[i - 1] + 1;
14     printf("%d ", magic_card[i]);
15     return print(n, i + 1, magic_card,cnt+1);
16 }
17 int main(void)
18 {
19     int magic_card[239000] = { 0 };
20     int i, j;
21     int n,x=0;
22     int cnt = 1,sum=0;
23     printf("몇장의 카드를 사용하시겠습니까?\n");
24     scanf("%d", &n);
25     printf("0~%d카드중에 숫자를 하나 고르세요(고른 후 엔터를 누르세요)\n", (int)pow(2, n) - 1);
26     for (i = 0; i < n; i++)
27     {
28         printf("%c카드 - 고른 숫자가 있으면 1 아니면 0을 입력하시오\n", 'A' + i);
29         magic_card[0] = (int)pow(2, i);
30         printf("%d ", magic_card[0]);
31         print(n, 1, magic_card, 1);
32         printf("\n");
33         scanf("%d", &x);
34         if (x == 1)
35             sum = sum + (int)pow(2, i);
36     }
37     printf("당신이 고른 숫자는 : %d 입니다.", sum);
38 }
```

2-3-3 디버깅 결과

```
C:\WINDOWS\system32\cmd.exe
몇장의 카드를 사용하시겠습니까?
4
0~15카드중에 숫자를 하나 고르세요(고른 후 엔터를 누르세요)
A카드 - 고른 숫자가 있으면 1 아니면 0을 입력하시오
1 3 5 7 9 11 13 15
0
B카드 - 고른 숫자가 있으면 1 아니면 0을 입력하시오
2 3 6 7 10 11 14 15
1
C카드 - 고른 숫자가 있으면 1 아니면 0을 입력하시오
4 5 6 7 12 13 14 15
1
D카드 - 고른 숫자가 있으면 1 아니면 0을 입력하시오
8 9 10 11 12 13 14 15
0
당신이 고른 숫자는 : 6 입니다.계속하려면 아무 키나 누르시오 . . .
```

2-3-4 재귀 호출의 오버플로우 발생

재귀 호출로 작성된 소스 코드에서 10장 이라는 숫자를 입력하였더니 다음과 같은 오버플로우가 발생을 하였다.



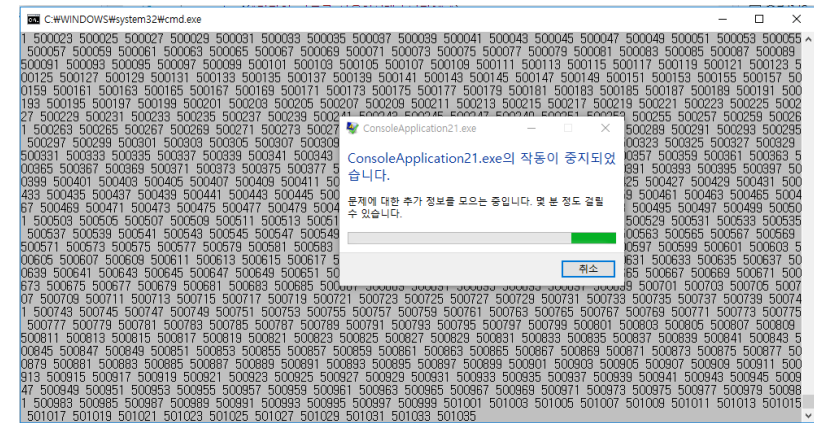
기존의 1차, 2차의 소스 같은 경우에는 15장의 카드를 입력을 하여도 문제없이 작동이 되었음을 확인을 하였다.

1차와 2차 소스에는 15장까지의 작동을 하였는데 재귀 호출 방식에서는 카드 10장에 오버플로우가 발생한 것을 보고 재귀 호출의 과도한 메모리 사용을 실감나게 경험을 해보았다. 메모리 측정은 정확한 측정은 아니지만 1.7mb가 측정되어 있었다. 앞에서 작성한 소스보다 확실히 메모리 사용량이 많은 점을 확인을 확인할 수 있었다.

2-4 문제해결 방식(4차 - 오버플로우의 극복)

2-4-1 오버플로우 발생 && 개선 방안

1차 소스에서 카드 몇 장을 입력했을 때 오버플로우가 발생하는지 테스트를 해보았다.



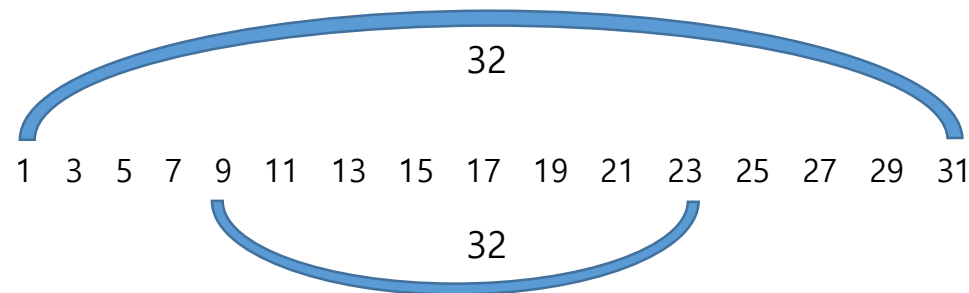
19장의 카드를 입력했을 때 다음과 같은 오버플로우가 발생하였다.

이번에는 오버플로우를 줄이는 방식으로 연구를 해보았다.

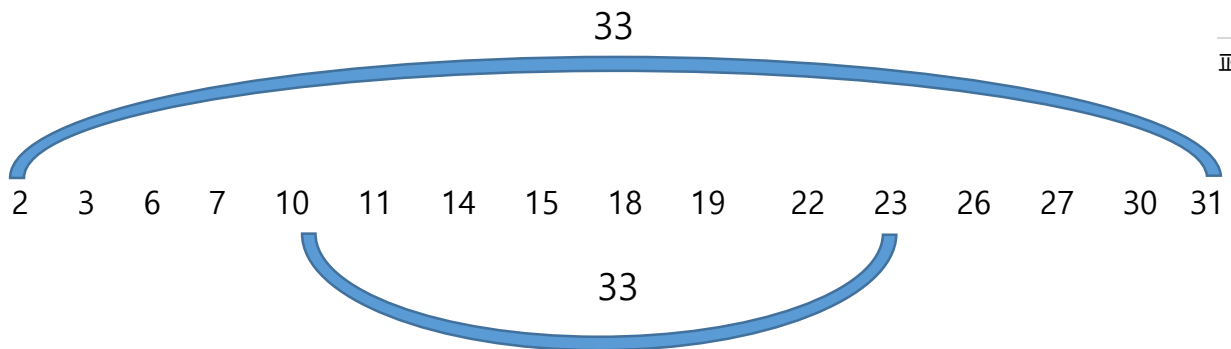
어떻게 하면 오버플로우를 줄일 수 있을까? 제일 일반적으로 떠올릴 수 있는 방법은 메모리의 크기를 늘리는 것이다. 메모리를 늘리는 것도 하나의 방법이지만 프로그래머는 어느 정도의 메모리의 절약과 확보를 할 줄 알아야 된다 생각한다.

하여 이번에 생각한 방식은 아래의 그림과 같다.

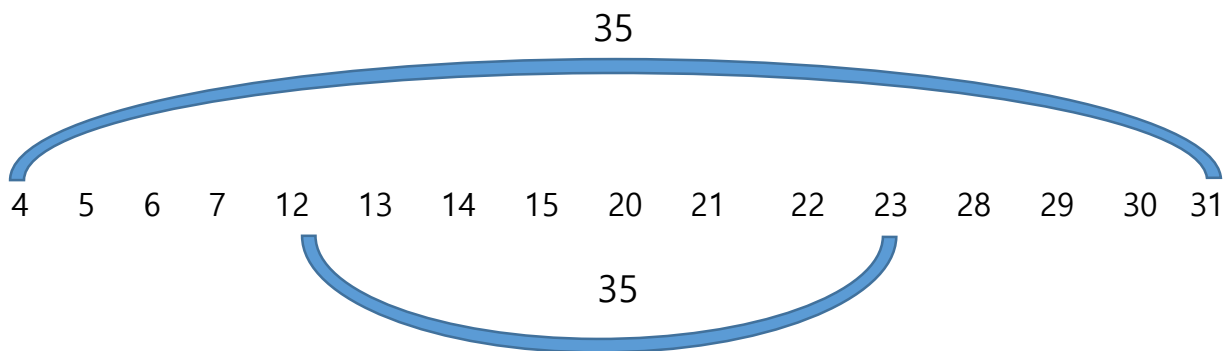
A카드 -



B카드 -



C카드 -



사진과 같이 앞의 숫자와 뒤에 숫자의 쌍의 합이 동일함과 수의 합은 2의 n승 + 첫번째 수 - 1과 동일함을 알 수가 있었다.

결론적으로 배열의 절반 까지만 알아도 모든 수의 값을 구하는 것이 가능하다는 것이다.

이번에는 위의 방식을 사용하여 프로그램을 작성을 해보았다.

2-4-2 소스 코드

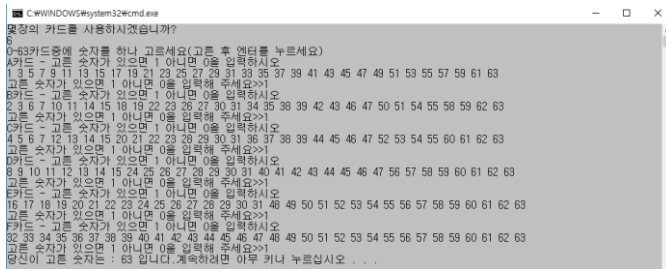
```
1  /*
2  20164006-박훈-마술카드(정적배열)-1차 개선 알고리즘 1차
3  */
4  #include<stdio.h>
5  #include<math.h>
6  int main(void)
7  {
8      unsigned long magic_card[239000] = { 0 };
9      int i, j;
10     int n;
11     int cnt = 1, sum = 0;
12     int x = 0;
13     printf("몇장의 카드를 사용하시겠습니까?\n");
14     scanf("%d", &n);
15     printf("0~%d카드중에 숫자를 하나 고르세요(고른 후 엔터를 누르세요)\n", (int)pow(2, n) - 1);
16     for (i = 0; i < n; i++)
17     {
18         printf("%c카드 - 고른 숫자가 있으면 1 아니면 0을 입력하시오\n", 'A' + i);
19         magic_card[0] = (int)pow(2, i);
20         printf("%d ", magic_card[0]);
21         cnt = 1;
22         for (j = 1; j < (int)pow(2, n - 1)/2; j++)
23         {
24             if (cnt == magic_card[0])
25             {
26                 magic_card[j] = magic_card[j - 1] + cnt + 1;
27                 cnt = 1;
28             }
29             else
30             {
31                 magic_card[j] = magic_card[j - 1] + 1;
32                 cnt++;
33             }
34             printf("%d ", magic_card[j]);
35         }
36         for (j=j-1; j >= 0; j--)
37         {
38             printf("%d ", (int)pow(2, n)+magic_card[0]-1 - magic_card[j]);
39         }
40         printf("\n고른 숫자가 있으면 1 아니면 0을 입력해 주세요>>");
41         scanf("%d", &x);
```

```

42     if (x == 1)
43         sum = sum + (int)pow(2, i);
44     }
45     printf("당신이 고른 숫자는 : %d 입니다.", sum);
46     return 0;
47 }

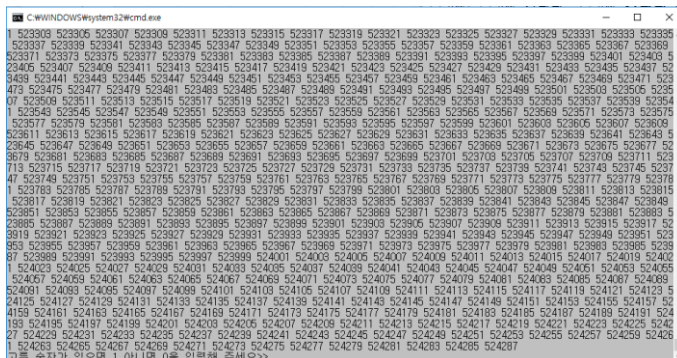
```

2-4-3 디버깅 결과

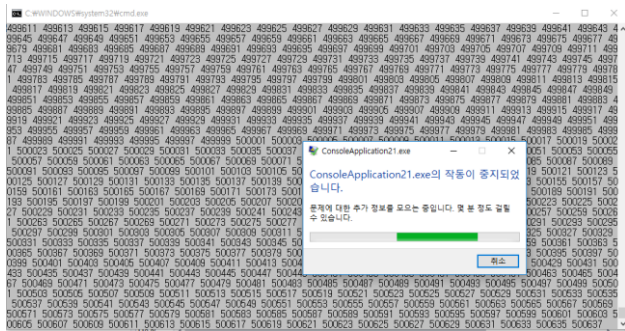


2-4-4 개선 결과 분석

19장을 입력했을 때의 결과입니다.



19장의 카드를 입력을 해도 문제없이 잘 작동할 수 있는 것을 볼 수가 있었습니다.



20장의 카드를 입력했을 때의 결과입니다. 20장의 카드를 입력했을 때 또다시 오버플로우가 발생하였습니다.

Ⅲ. 결론

1. 1단계 최종 결과

1-1. 최종 소스

```
1  /*
2  20164006-박훈-마술카드(정적배열)-1차 개선 알고리즘 1차
3  */
4  #include<stdio.h>
5  #include<math.h>
6  int main(void)
7  {
8      unsigned long magic_card[239000] = { 0 };
9      int i, j;
10     int n;
11     int cnt = 1, sum = 0;
12     int x = 0;
13     printf("몇장의 카드를 사용하시겠습니까?\n");
14     scanf("%d", &n);
15     printf("0~%d카드중에 숫자를 하나 고르세요(고른 후 엔터를 누르세요)\n", (int)pow(2, n) - 1);
16     for (i = 0; i < n; i++)
17     {
18         printf("%c카드 - 고른 숫자가 있으면 1 아니면 0을 입력하시오\n", 'A' + i);
19         magic_card[0] = (int)pow(2, i);
20         printf("%d ", magic_card[0]);
21         cnt = 1;
22         for (j = 1; j < (int)pow(2, n - 1)/2; j++)
23         {
24             if (cnt == magic_card[0])
25             {
26                 magic_card[j] = magic_card[j - 1] + cnt + 1;
27                 cnt = 1;
28             }
29             else
30             {
31                 magic_card[j] = magic_card[j - 1] + 1;
32                 cnt++;
33             }
34             printf("%d ", magic_card[j]);
```

```
35     }
36     for (j=j-1; j >= 0; j--)
37     {
38         printf("%d ", (int)pow(2, n)+magic_card[0]-1 - magic_card[j]);
39     }
40     printf("\n고른 숫자가 있으면 1 아니면 0을 입력해 주세요>>");
41     scanf("%d", &x);
42     if (x == 1)
43         sum = sum + (int)pow(2, i);
44     }
45     printf("당신이 고른 숫자는 : %d 입니다.", sum);
46     return 0;
47 }
```

1-2. 1차 레포트를 마치며 느낀 점

비트 연산의 곱셈의 결과가 저의 예상과 다른 것을 보며 머리 속에 당연히 그럴 것이다. 이러한 판단이 잘못되었음을 알았습니다. 프로그램은 구상하는 것도 중요하지만 직접 프로그램을 코딩을 직접 작성해 보아야 한다는 것을 알게 되었습니다. 또한 1학년때 c실습으로 많은 코딩 문제를 풀었지만 디버깅 결과만 가지고 판단했던 것에 이번 레포트를 작성하면서 많은 부끄러움을 느꼈습니다. 재가 코딩한 소스를 분석하는 습관을 기르도록 노력하겠습니다. 이러한 마음가짐을 가지고 다음 단계에도 열심히 도전해 보도록 하겠습니다.