EC C언어 스터디

-4강-

에라토스테네스의 체

정렬 알고리즘

구간 합

시공간 복잡도

배열에서 n개 고르기

다차원 배열

시공간 복잡도

시공간 복잡도

- 문제의 시간제한, 메모리제한, 입력 데이터의 크기를 보고
  얼마의 시공간 복잡도 내에 풀어야 할 지 파악하는 것

- 코드를 보고 얼마의 시공간 복잡도를 가질지 파악하는 것

시간 복잡도 = 입력의 크기에 따른 연산량

Big-O 표기법

- 시공간 복잡도

- O(연산량)으로 표기
- 항상 최악의 경우를 생각한다

O(n) = 입력의 크기가 n일 때 최대 n번의 연산을 한다.

<배열의 합>

```c
#include <stdio.h>

int main()
{
    int n;
    scanf("%d", &n);1
    for(int i=0;i<n;i++) scanf("%d",&arr[i]); n

    int sum=0;
    for(int i=0;i<n;i++) sum+=arr[i]; n
}
```

O(2n+1) => O(n)

<최대공약수 알고리즘>

```c
#include <stdio.h>

int main()
{
	int a, b;
	scanf("%d%d", &a, &b);
	int c = (a > b ? b : a);
	int gcd;
	for (int i = c; i > 0; i--)
	{
		if (a%i == 0 && b%i == 0)
		{
			gcd = i;
			break;
		}
	}
	printf("%d", gcd);
}
```

O(b) or O(a) => O(n)

<배열의 합>

O(1)

```c
#include <stdio.h>

int main()
{
    int n;
    scanf("%d", &n);
    n++;
    printf("입력값+1 = %d", n);
}
```

<수 뒤집기>

```c
#include <stdio.h>

int main()
{
	int n;
	scanf("%d", &n);

	int rev = 0;
	while (n > 0)
	{
		rev *= 10;
		rev += n % 10;
		n /= 10;
	}
	printf("%d", rev);
}
```

$O(log_{10}n)$

<이진수 구하기>

$$O(log_2 n + log_{10} n) => O(log_2 n)$$

$$O(\log n)$$

```c
#include <stdio.h>

int main()
{
	int n;
	scanf("%d", &n);

	int bi = 0;
	while (n > 0)
	{
		bi *= 10;
		bi += n % 2;
		n /= 2;
	}

	// 이진수가 거꾸로 저장되기 때문에 뒤집어줘야
	int rev = 0;
	while (bi > 0)
	{
		rev *= 10;
		rev += bi % 10;
		bi /= 10;
	}

	printf("%d", rev);
}
```

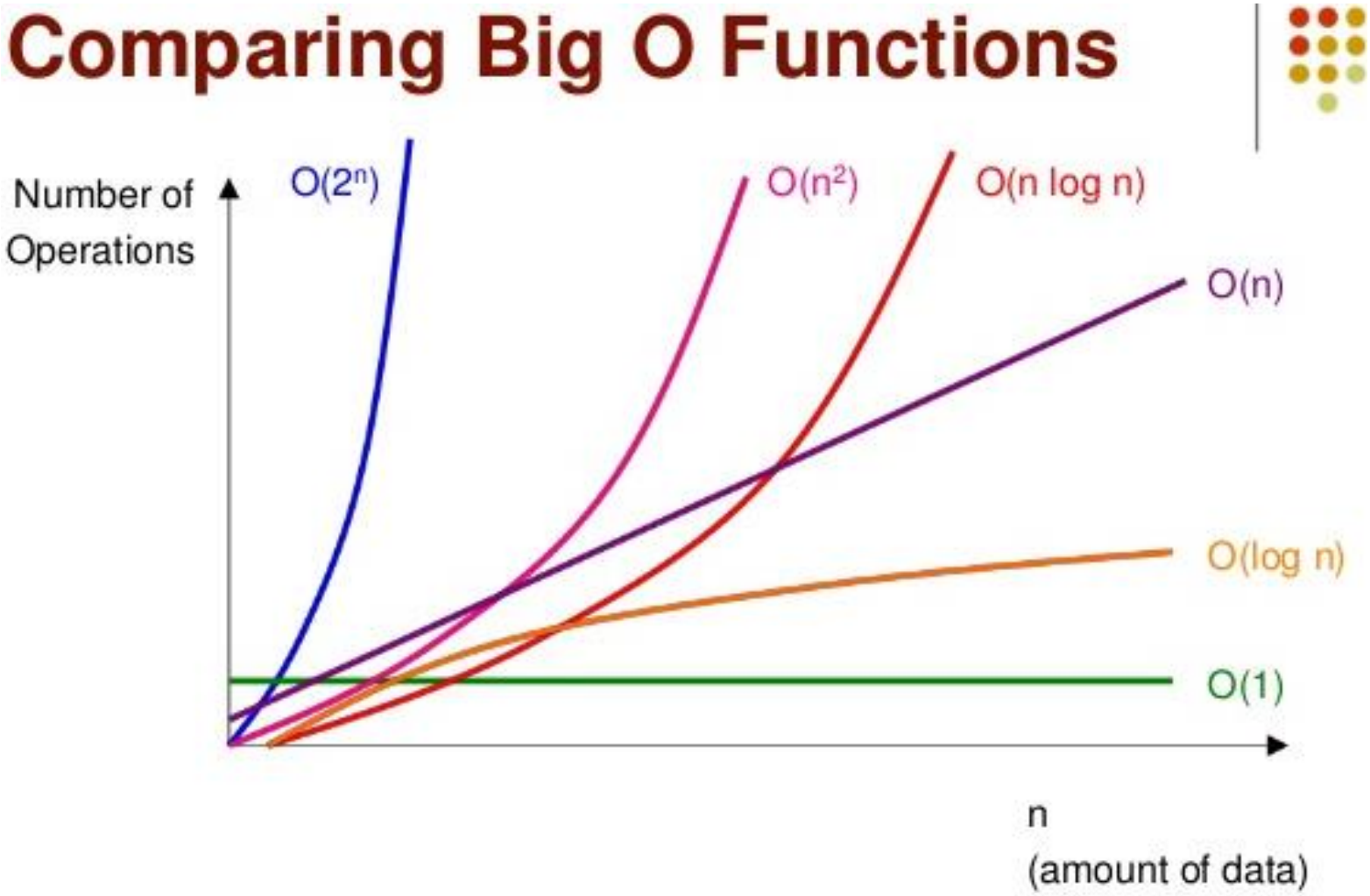
<버블 소트>

$O(n(n-1)/2) => O(n^2)$

```c
#include <stdio.h>

int main()
{
	int arr[5] = { 3,2,1,4,5 };

	for (int i = 0; i < 5; i++)
	{
		for (int j = i + 1; j < 5; j++)
		{
			if (arr[i] > arr[j])
			{
				int temp = arr[i];
				arr[i] = arr[j];
				arr[j] = temp;
			}
		}
	}

	for (int i = 0; i < 5; i++) printf("%d ", arr[i]);
}
```

Big-O 표기법

- 시공간 복잡도



프로그램이 실행되면서 메모리를 차지하는 정도

<X보다 작은 수>

```c
#include <stdio.h>

int main()
{
    int n, m;
    scanf("%d%d",&n,&m);
    for(int i=0;i<n;i++)
    {
        int temp;
        scanf("%d",&temp);
        if(temp < m) printf("%d ",temp);
    }
}
```

<X보다 작은 수>

```c
#include <stdio.h>

int main()
{
	int n, m;
	scanf("%d%d", &n, &m);
	int arr[10001];
	for (int i = 0; i < n; i++)
	{
		scanf("%d", &arr[i]);
		if (arr[i] < m) printf("%d ", arr[i]);
	}
}
```

배열의 크기가 커지면??

에라토스테네스의 체

<X보다 작은 수>

```c
#include <stdio.h>

int main()
{
	int n, check = 1;
	scanf("%d", &n);
	for (int i = 2; i < n; i++)
	{
		if (n%i == 0)
		{
			check = 0;
			break;
		}
	}
	if (n <= 1) check = 0;
	if (check) printf("소수입니다.");
	else printf("소수가 아닙니다.");
}
```

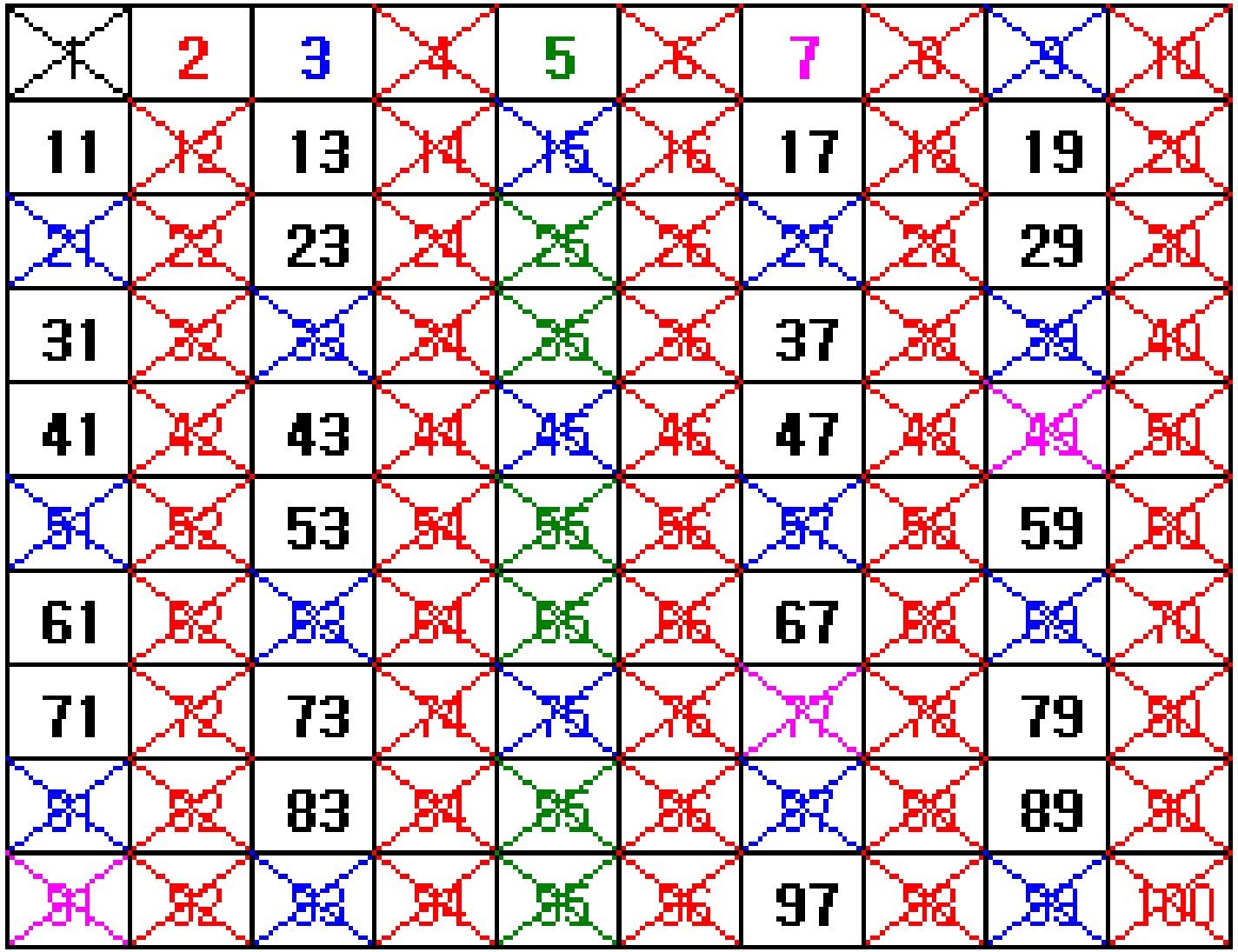
O(n)

에라토스테네스의 체

- O(1)만에 소수를 판별할 수 있다!
- 소수의 배수들은 소수가 아님을 이용한 알고리즘

에라토스테네스의 체

에라토스테네스의 체



에라토스테네스의 체

<에라토스테네스의 체>

```c
#include <stdio.h>

int main()
{
	int prime[10000] = { 1,1, }; //0이면 소수
	for (int i = 2; i < 10000; i++)
	{
		if (prime[i]) continue;
		for (int j = 2; i*j < 10000; j++)
		{
			prime[i*j] = 1;
		}
	}

	for (int i = 0; i < 10000; i++)
	{
		if (!prime[i]) printf("%d ", i);
	}
}
```

1. 원하는 크기의 배열을 만든다
2. 2부터 시작하여 처음 만나는 수는 모두 소수이다
3. 처음 만나는 수의 배수들은 모두 소수가 아니고 한번 만난 것으로 간주한다.
4. 배열의 값이 0이면 소수인 것이다.

만능인 것처럼 보이지만 공간복잡도의 문제가..

풀어보세요!

- 소수 구하기 : O(n^2)으론 풀 수 없습니다.

배열에서 n개 고르기

- n중첩 반복문 사용
- 재귀함수 사용

```c
//1개씩 고르기
for (int i = 0; i < 5; i++) printf("%d\n", arr[i]);
```

```
//2개씩 고르기
for (int i = 0; i < 5; i++)
{
        for (int j = i + 1; j < 5; j++)
        {
                printf("%d %d\n", arr[i], arr[j]);
        }
}
```

- i=0, j=1,2,3,4
- i=1, j=2,3,4
- i=2, j=3,4
- i=3, j=4

3개 고르기

```
//3개씩 고르기
for (int i = 0; i < 5; i++)
{
	for (int j = i + 1; j < 5; j++)
	{
		for (int k = j + 1; k < 5; k++)
		{
			printf("%d %d %d\n", arr[i], arr[j], arr[k]);
		}
	}
}
```

- i=0, j=1, k=2,3,4
- i=0, j=2, k=3,4
- i=0, j=3, k=4
- i=1, j=2, k=3,4
- i=1, j=3, k=4
- i=2, j=3, k=4

풀어보세요!

- 로또 : 배열에서 '6'개를 고릅니다!

정렬 알고리즘

버블 소트

```c
#include <stdio.h>

int main()
{
        int arr[5] = { 3,2,1,4,5 };

        for (int i = 0; i < 5; i++)
        {
                for (int j = i + 1; j < 5; j++)
                {
                        if (arr[i] > arr[j])
                        {
                                int temp = arr[i];
                                arr[i] = arr[j];
                                arr[j] = temp;
                        }
                }
        }

        for (int i = 0; i < 5; i++) printf("%d ", arr[i]);
}
```

원소의 쌍을 모두 비교

swap

- i=0, j=1,2,3,4
- i=1, j=2,3,4
- i=2, j=3,4
- i=3, j=4

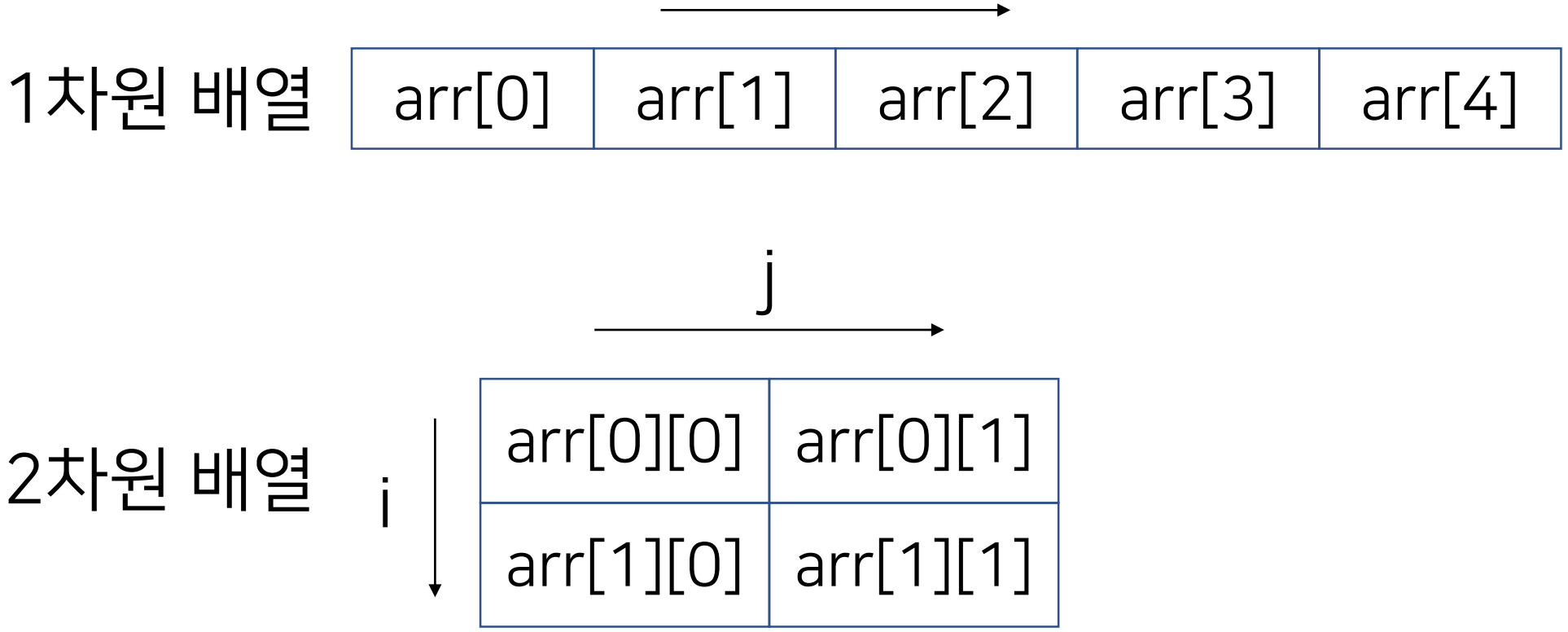
풀어보세요!

- 소트인사이드 : 문자열을 정렬해봅시다.

다차원 배열

- 인덱스가 여러개인 배열
- 2차원 배열 -> 표처럼 다뤄짐
- 3차원 배열 -> 큐브같은 모양
- n차원 배열은 n중 for문으로 다룬다

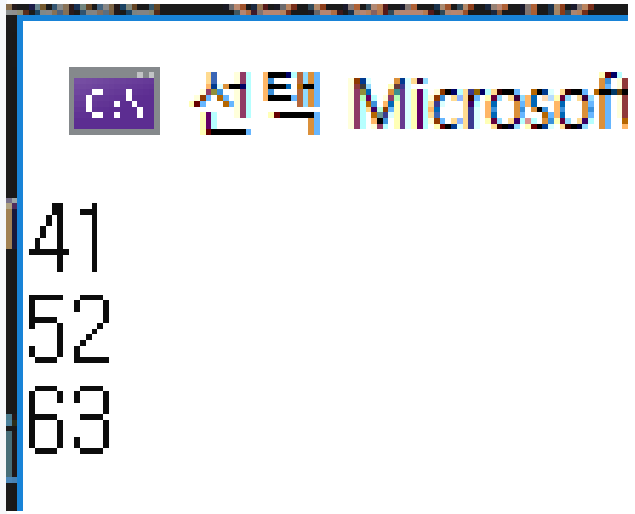
다차원 배열


1차원 배열
arr[0]
arr[1]
arr[2]
arr[3]
arr[4]
j
2차원 배열
i
arr[0][0]
arr[0][1]
arr[1][0]
arr[1][1]

```c
#include <stdio.h>

int main()
{
	int arr[5][5];
	for (int i = 0; i < 5; i++)
	{
		for (int j = 0; j < 5; j++)
		{
			scanf("%d", &arr[i][j]);
		}
	}
	for (int i = 0; i < 5; i++)
	{
		for (int j = 0; j < 5; j++)
		{
			printf("%d", arr[i][j]);
		}
	}
}
```

2차원 배열

```c
#include <stdio.h>

int main()
{
	int arr2[2][3] =
	{
		{1,2,3},
		{4,5,6},
	};

	for (int j = 0; j < 3; j++)
	{
		for (int i = 1; i >= 0; i--)
		{
			printf("%d", arr2[i][j]);
		}
		printf("\n");
	}
}
```



풀어보세요!

- 하얀 칸 : 2차원 배열 다루기

구간 합

- 배열에서 어떤 구간 내부의 원소들의 합을 구하는 것
- 반복문을 통해서 구할 수 있으나 구간 합을 여러 번 구해야 할 때는 더 효율적인 방법을 사용해야 한다

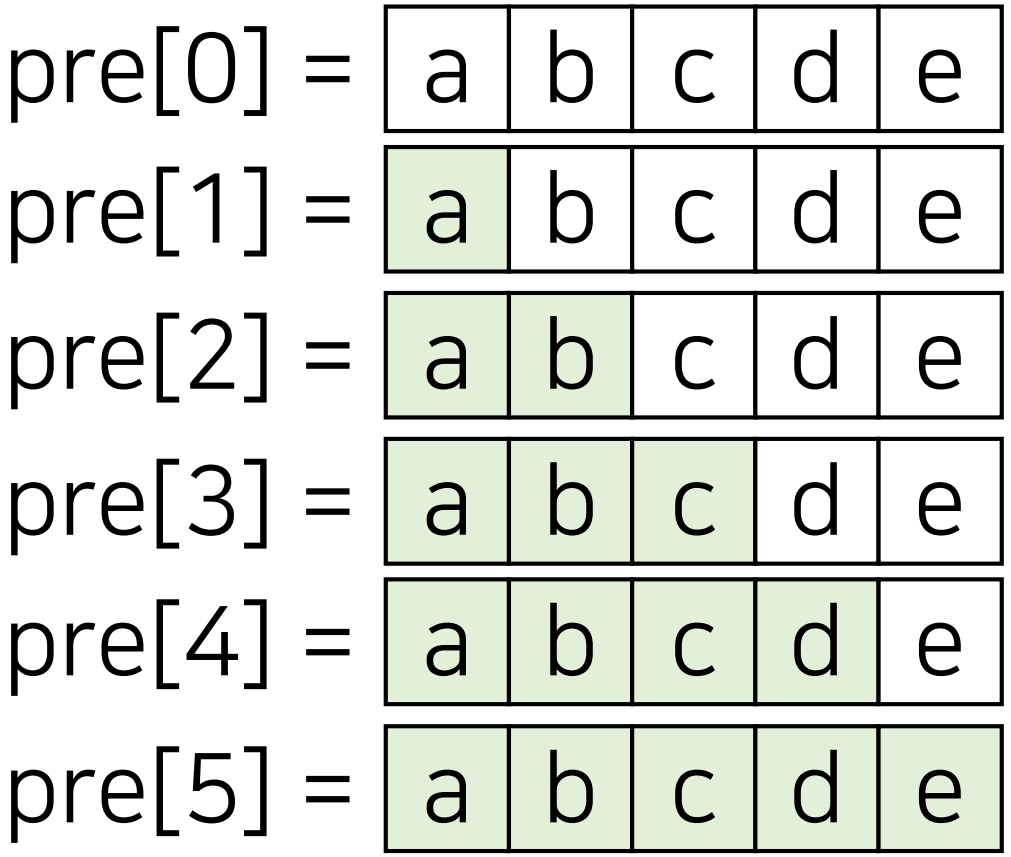
구간 합(누적합 이용)

- arr[5] = {a, b, c, d, e}
- pre[6] = {0, a, a+b, a+b+c, a+b+c+d, a+b+c+d+e}

arr의 2번 째 원소부터 4번 째 원소 까지의 합
= b+c+d = pre[4]-pre[1]

O(n) O(1)

구간 합(누적합 이용)



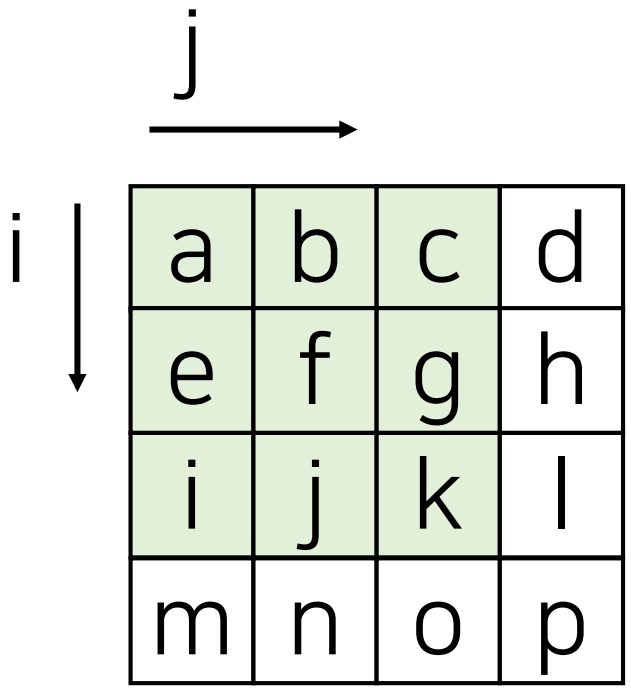
2번째~4번째 합 = pre[4]-pre[1]

(b+c+d)

구간 합(누적합 이용)

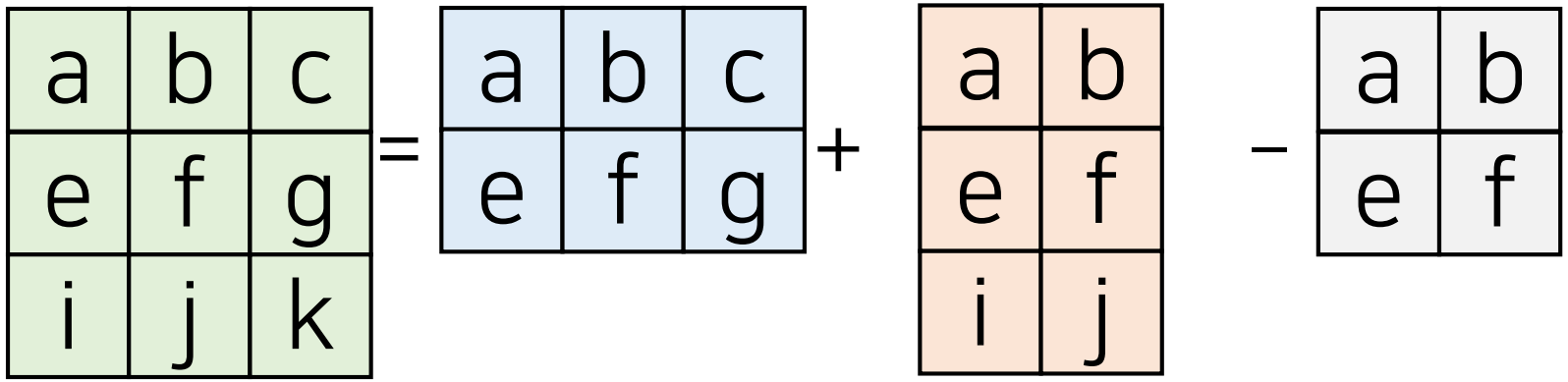

```c
#include <stdio.h>

int main()
{
	int arr[] = { 1,2,3,4,5 };

	int pre[6] = {0,};
	for (int i = 0; i < 5; i++) pre[i + 1] = arr[i] + pre[i];
	int a, b;
	scanf("%d%d", &a, &b);
	printf("구간 [%d, %d]의 합 = %d", a, b, pre[b] - pre[a - 1]);
}
```

2차원 배열에서 구간합

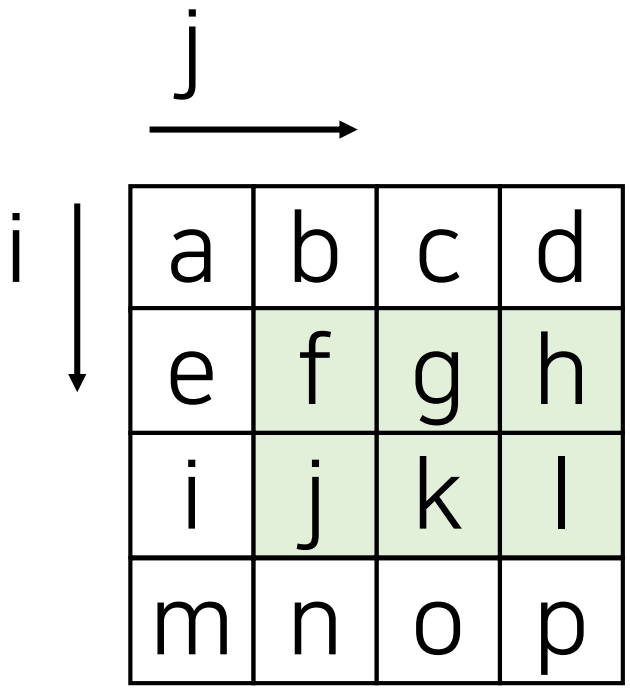


pre[5][5] = 자신보다 인덱스가 작은 arr값의 누적합

pre[3][3] = pre[2][3] + pre[3][2] – pre[2][2]

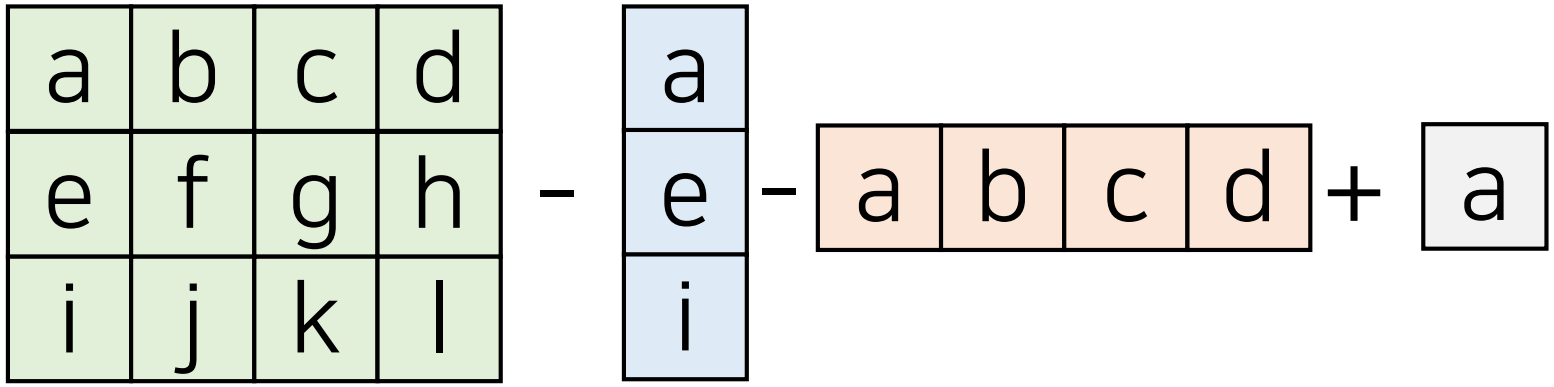


2차원 배열에서 구간합



<arr[4][4]>

(3, 4)~(2, 2) = pre[3][4] - pre[3][1] – pre[1][4] + pre[1][1]



2차원 배열에서 구간합


```c
#include <stdio.h>

int main()
{
	int arr[5][5] =
	{
		{1,2,3,4,5},
		{5,4,3,2,1},
		{3,4,5,6,7},
		{1,1,1,1,1},
		{2,2,2,2,2},
	};

	int pre[6][6] = { 0, };
	for (int i = 0; i < 5; i++)
	{
		for (int j = 0; j < 5; j++)
		{
			pre[i + 1][j + 1] = arr[i][j] + pre[i][j + 1] + pre[i + 1][j] - pre[i][j];
		}
	}

	int a, b, c, d;
	scanf("%d%d%d%d", &a, &b, &c, &d);
	printf("(%d, %d) 부터 (%d, %d) 까지의 합 = %d",
		a, b, c, d, pre[c][d] - pre[a - 1][d] - pre[c][b - 1] + pre[a - 1][b - 1]);
}
```

풀어보세요!

- 구간 합 구하기 4 : O(nm)으론 풀 수 없습니다.
- 구간 합 구하기 5 : O(n^2m)으론 풀 수 없습니다.

수고하셨습니다!