01. Complexity

Div. 3 알고리즘 스터디 / 임지환

알고리즘을 공부하는 이유?

- 같은 문제일지라도 문제를 해결하는 알고리즘은 여러 종류
- 한정된 자원 안에서 최적의 선택이 요구됨
- 어떠한 기준으로 적절한 알고리즘을 선택할 수 있을까?

Complexity Analysis

Complexity Analysis

Time

- 데이터 값이 커짐에 따라 수행시간이 필연적으로 증가.
- 이 수행시간이 느리게 증가하는 알고리즘이 더 효율적이라 볼 수 있다.

Space

- 제한된 메모리에서 실행될 수 있게끔 하는 알고리즘이 필요할 수도 있다.

시간 복잡도 분석

- 그냥 걸리는 시간을 측정한다면?

```cpp
int main() {

    int *arr = (int*)malloc(sizeof(int) * 100000000);
    for (int i = 0; i < 100000000; i++)
        arr[i] = i + 1;
    int tofind = 100000000, findidx = -1;
    std::chrono::system_clock::time_point StartTime = std::chrono::system_clock::now();

    binary_search(arr, 100000000, tofind, &findidx);

    std::chrono::system_clock::time_point EndTime = std::chrono::syste

    std::chrono::duration<double> DefaultSec = EndTime - StartTime;
    std::chrono::nanoseconds nano = EndTime - StartTime;
    if (findidx != -1) {
        printf("key value is in array of %dth.\n", findidx);
    }
    else printf("not found\n");
    std::cout << "Due time : " << nano.count() << "ns" << '\n';
    free(arr);
    return 0;
}
```

C:₩WINDOWS₩system32₩cmd.exe

```
key value is in array of 99999999th.
Due time : 4400ns
계속하려면 아무 키나 누르십시오 . . .
```

시간 복잡도 분석

- 그냥 걸리는 시간을 측정한다면?

```cpp
int main() {

    int *arr = (int*)malloc(sizeof(int) * 100000000);
    for (int i = 0; i < 100000000; i++)
        arr[i] = i + 1;
    int tofind = 100000000, findidx = -1;
    std::chrono::system_clock::time_point StartTime = std::chrono::system_clock::now();

    binary_search(arr, 100000000, tofind, &findidx);

    std::chrono::system_clock::time_point EndTime = std::chrono::syste

    std::chrono::duration<double> DefaultSec = EndTime - StartTime;
    std::chrono::nanoseconds nano = EndTime - StartTime;
    if (findidx != -1) {
        printf("key value is in array of %dth.\n", findidx);
    }
    else printf("not found\n");
    std::cout << "Due time : " << nano.count() << "ns" << '\n';
    free(arr);
    return 0;
}
```

C:\WINDOWS\system32\cmd.exe

```
key value is in array of 99999999th.
Due time : 4400ns
계속하려면 아무 키
```

C:\WINDOWS\system32\cmd....

```
key value is in array of 99999999th.
Due time : 2000ns
계속하려면 아무 키나 누르십시오 . . .
```

???

연산 횟수 측정

ex) 1~n까지의 합

$$\sum_{i=1}^{n} i$$

연산 횟수 측정

ex) 1~n까지의 합

$$\sum_{i=1}^{n} i$$

```
int sum = 0;
for(int i = 1; i<= n; i++)
    sum += i;
```

연산 횟수 측정

ex) 1~n까지의 합

$$\sum_{i=1}^{n} i$$

```
int sum = 0;
for(int i = 1; i<= n; i++)
    sum += i;
```

ret변수 선언 및 초기화

변수 i 선언 및 초기화

i<=n 연산 (n+1)번
i++ 연산 n번
ret += I 연산 n번
…

약 (3n + 3)번

연산 횟수 측정

ex) 1~n까지의 합

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

```
int sum = 0;
for(int i = 1; i<= n; i++)
    sum += i;
```

연산 횟수 측정

ex) 1~n까지의 합

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

```
int sum = 0;
for(int i = 1; i<= n; i++)
    sum += i;
```

```
int sum = (n * (n+1)) / 2;
```

연산 횟수 측정

ex) 1~n까지의 합

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

```
int sum = 0;
for(int i = 1; i<= n; i++)
    sum += i;
```

```
int sum = (n * (n+1)) / 2;
```

3번!

Example 1

오름차순 정렬

5	2	3	4	1

→

1	2	3	4	5

Bubble Sort

5	2	3	4	1

Bubble Sort

5	2	3	4	1

Bubble Sort

5	2	3	4	1

2	5	3	4	1

Bubble Sort

5	2	3	4	1

2	5	3	4	1

Bubble Sort

5	2	3	4	1

2	5	3	4	1

2	3	5	4	1

Bubble Sort

5	2	3	4	1

2	5	3	4	1

2	3	5	4	1

Bubble Sort

5	2	3	4	1

2	5	3	4	1

2	3	5	4	1

2	3	4	5	1

Bubble Sort

5	2	3	4	1
2	5	3	4	1
2	3	5	4	1
2	3	4	5	1

Bubble Sort

5	2	3	4	1

2	5	3	4	1

2	3	5	4	1

2	3	4	5	1

2	3	4	1	5

Bubble Sort

5	2	3	4	1

2	5	3	4	1

2	3	5	4	1

2	3	4	5	1

2	3	4	1	5

4번의 swap 발생

Bubble Sort

2	3	4	1	5

Bubble Sort

2	3	4	1	5

Bubble Sort

2	3	4	1	5

2	3	1	4	5

Bubble Sort

2	3	4	1	5
2	3	1	4	5

→ 1번의 swap 발생

Bubble Sort

2	3	1	4	5

Bubble Sort

2	3	1	4	5

Bubble Sort

2	3	1	4	5

2	1	3	4	5

Bubble Sort

2	3	1	4	5

2	1	3	4	5

→ 1번의 swap 발생

Bubble Sort

2	1	3	4	5

Bubble Sort

Bubble Sort

2	1	3	4	5

1	2	3	4	5

Bubble Sort

2	1	3	4	5

1	2	3	4	5

→ 1번의 swap 발생

```
void bubble_sort(int* arr, int n) {
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j+1]) {
                int tmp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = tmp;
            }
}
```

```
void bubble_sort(int* arr, int n) {
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j+1]) {
                int tmp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = tmp;
            }
}
```

최외곽 for문이 (n-1)번 실행

```c
void bubble_sort(int* arr, int n) {
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j+1]) {
                int tmp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = tmp;
            }
}
```

다음 for문이 (n-i-1)번 실행

```
void bubble_sort(int* arr, int n) {
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j+1]) {
                int tmp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = tmp;
            }
}
```

조건에 따라 시행될 수도,
안될 수도 있음

```
void bubble_sort(int* arr, int n) {
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j+1]) {
                int tmp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = tmp;
            }
}
```

$$\sum_{i=0}^{n-2} \sum_{j=0}^{n-i-1} (conditional)$$

```
void bubble_sort(int* arr, int n) {
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j+1]) {
                int tmp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = tmp;
            }
}
```

$$\sum_{i=0}^{n-2} \sum_{j=0}^{n-i-1} (conditional)$$

$$= \sum_{i=0}^{n-2} (n-i-1)$$

$$= \sum_{i=0}^{n-2} n - \sum_{i=0}^{n-2} i - \sum_{i=0}^{n-2} 1$$

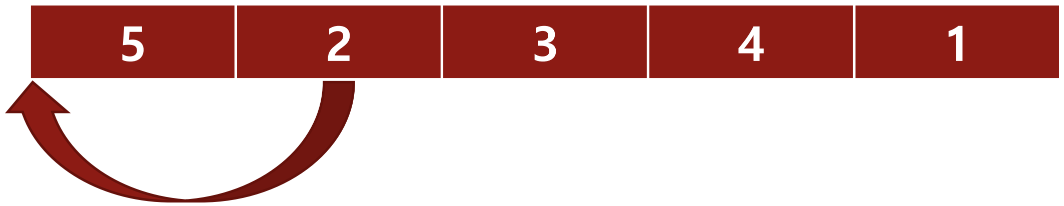
$$= n(n-1) - \frac{n(n-1)}{2} - (n-1)$$

$$(n-1)(n - \frac{n}{2} - 1) = \frac{(n-1)(n-2)}{2}$$

Insertion Sort

5	2	3	4	1

Insertion Sort



Insertion Sort

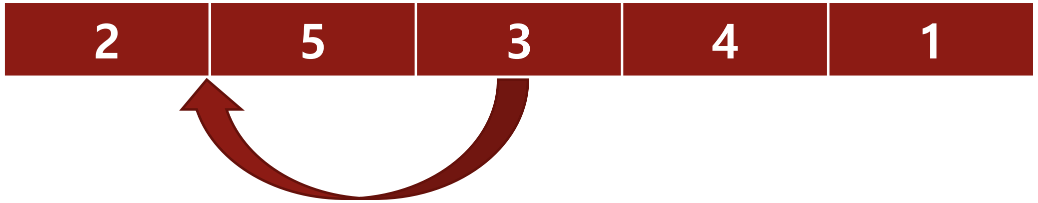
5	2	3	4	1

2	5	3	4	1

Insertion Sort

2	5	3	4	1

Insertion Sort



Insertion Sort

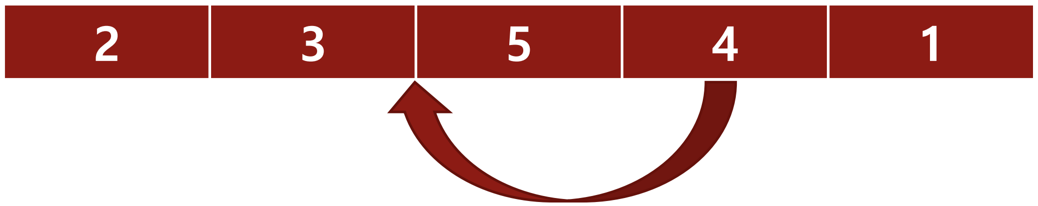
2	5	3	4	1

2	3	5	4	1

Insertion Sort

2	3	5	4	1

Insertion Sort



Insertion Sort

2	3	5	4	1

2	3	4	5	1

Insertion Sort

2	3	4	5	1

Insertion Sort

Insertion Sort

2	3	4	5	1

1	2	3	4	5

```
void insertion_sort(int* arr, int n) {
    for (int i = 1; i < n; i++) {
        int tmp = arr[i], j = i;
        while (--j >= 0 && tmp < data[j]) {
            arr[j+1] = arr[j];
            arr[j] = tmp;
        }
    }
}
```

```
void insertion_sort(int* arr, int n) {
    for (int i = 1; i < n; i++) {
        int tmp = arr[i], j = i;
        while (--j >= 0 && tmp < data[j]) {
            arr[j+1] = arr[j];
            arr[j] = tmp;
        }
    }
}
```

최외곽 for문이 (n-1)번 실행

최악의 경우, j가 i번 이동

```
void insertion_sort(int* arr, int n) {
    for (int i = 1; i < n; i++) {
        int tmp = arr[i], j = i;
        while (--j >= 0 && tmp < data[j]) {
            arr[j+1] = arr[j];
            arr[j] = tmp;
        }
    }
}
```

$$\sum_{i=1}^{n-1} \sum_{j=0}^{i-1} (conditional)$$

$$= \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$$

Bubble sort Vs. Insertion sort

- 두 경우 모두 조건에 따라 연산이 생략되는 경우가 생길 수 있다.
- But, **최악의 경우**, 두 방법 다 연산횟수가 n^2에 비례함을 볼 수 있다.

Example 2

탐색

```
int linear_search (int* arr, int size, int key) {
    int ret = -1;
    for (int i = 0; i < size; i++)
        if (arr[i] == key) {
            ret = i;
            return ret;
        }
    return ret;
}
```

배열의 맨 처음부터 끝까지 탐색을 하는데
만약 찾고자 하는 값이 없다면?
: 최악의 경우 size만큼의 탐색 실행

탐색 시간 $\propto$ size

Example 2

이분 탐색

```
int binary_search (int* arr, int size, int key) {
    int left = 0, right = size – 1, ret = -1;
    while (left <= right) {
        int mid = (left + right) / 2;
        if (arr[mid] == key) {
            return ret = mid;
        }
        if (arr[mid] > key)
            right = mid  1;
        else left = mid + 1;
    }
    return ret;
}
```

Example 2

이분 탐색

```
int binary_search (int* arr, int size, int key) {
    int left = 0, right = size – 1, ret = -1;
    while (left <= right) {
        int mid = (left + right) / 2;
        if (arr[mid] == key) {
            return ret = mid;
        }
        if (arr[mid] > key)
            right = mid  1;
        else left = mid + 1;
    }
    return ret;
}
```

탐색에 성공하든 실패하든 다음 탐색 범위가 절반씩 줄어듬.

$$T(n) = T(\frac{n}{2}) + c, T(1) = c$$

$$let\ n = 2^m,$$
$$T(n) = T(2^m) = T(2^{m-1}) + c$$
$$T(2^{m-1}) = T(2^{m-2}) + c$$
$$thus,\ T(2^m) = T(2^{m-2}) + c + c$$
$$= T(2^{m-3}) + c + c + c = T(2^{m-m}) + mc$$
$$= T(2^0) + cm = T(1) + cm$$
$$= c + cm = c(1+m)$$

$$\text{since}\ \ m = \log_2 n,$$
$$T(n) = c(1+m) = c(1+\log_2 n) \propto \log_2 n$$

객관적인 성능 분석법

1) 구체적인 시간 측정보다는 연산 횟수 측정

: 시간 측정은 컴퓨터의 성능에 따라 달라질 수 있는 부분이다.

2) 임의의 경우에 대한 측정보다는 최악의 경우를 고려

: 같은 형태의 데이터에 대해 다른 알고리즘의 성능을 봐야 하므로

Big-O notation

- 계수 & 낮은 차수의 항 제외

  -> 점근적 묘사(asymptotic description; as upper bound)

- Ex:
  - $37\log n + 0.1n = O(n)$
  - $n^2 + 10n = O(n^2)$
  - $4(\log n)^2 + n\log n + 100n = O(n\log n)$
  - $n^2 + 10n = O(n^{200})$
  - …

앞에서 봤던 Sort & Search Algorithm?

- Bubble sort :
- Insertion sort :
- Linear search :
- Binary search :

Big-O notation

- 계수 & 낮은 차수의 항 제외

  -> 점근적 묘사(asymptotic description; as upper bound)

– Ex:
  - $37\log n + 0.1n = O(n)$
  - $n^2 + 10n = O(n^2)$
  - $4(\log n)^2 + n\log n + 100n = O(n\log n)$
  - $n^2 + 10n = O(n^{200})$
  - $\ldots$

앞에서 봤던 Sort & Search Algorithm?

- Bubble sort : $O(n^2)$
- Insertion sort : $O(n^2)$
- Linear search : $O(n)$
- Binary search : $O(logn)$

Example 3

```
void swap (int *A, int *B) {
    int tmp = *A;
    *A = *B;
    *B = tmp;
}
```

Example 3

```
void swap (int *A, int *B) {
    int tmp = *A;
    *A = *B;
    *B = tmp;
}
```

데이터의 크기가 늘어나더라도,
swap함수는 두 데이터에 대해서만 실행

시간복잡도 : $O(1)$

주로 만날 수 있는 시간복잡도

$$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3)$$
$$< O(2^n) < O(n!) < O(n^n) < \ \ldots.$$