과제

장유선

2023.08.25

1. 문제 정의

삽입 정렬 알고리즘을 구현해 보자

구현 방법

1. 정렬이 완료 된 Index를 지정, 처음은 0
2. 정렬을 수행 할 Index를 지정, 처음은 1
3. 정렬을 수행 할 Index가 배열의 길이가 될 때 까지 아래 작업을 반복
    a. 정렬이 완료 된 Index와 값을 비교한다
        i. 정렬을 수행 할 Index가 더 큰 값인 경우, 정렬을 수행 할 Index를 1 증가
        ii. 정렬을 수행 할 Index가 더 작은 경우 앞으로 탐색하면서 작거나 같은 값을 찾고, 해당 다음 Index에 비교 값을 삽입한다
        그리고 정렬이 완료된 Index, 정렬을 수행 할 Index를 각각 1씩 증가한다

테스트 방법

1. 임의의 100,000개의 값을 지니는 List or 연결리스트를 만들고 해당 값의 정렬 시간 측정
2. 1 ~ 100,000까지 값을 지니는 List or 연결리스트를 만들고 해당 값의 정렬 시간 측정

2. 개념 설명

삽입 정렬은 두 번째 원소부터 시작하여 그 앞의 원소들과 비교하여 삽입할 위치를 지정한 후, 원소를 뒤로 옮기고 지정된 자리에 자료를 삽입하여 정렬하는 알고리즘이다.

모든 원소가 이미 정렬이 되어있는 경우, 외부 루프를 N-1번 도는 동안 비교 연산은 1번씩 수행된다. 따라서 최선의 경우, Best T(n) = (N-1)*1

O(n) = n 이 된다.

모든 원소가 역순으로 정렬되어 있는 경우, 외부 루프를 N-1번 도는 동안 비교연산은 1, 2, … , (N-1)번 수행된다. 따라서 최악의 경우, Worst T(n) = 1+2+…+(N-1) = (N-1)*N/2

O(n) = n^2 이 된다.

3. Python Code Hard Copy

```python
def insertion_sort(arr):
    n = len(arr)

    for i in range(1, n):
        temp = arr[i]
        j = i - 1

        while j >= 0 and arr[j] > temp:
            arr[j + 1] = arr[j]
            j -= 1

        arr[j + 1] = temp

        # 정렬 과정을 출력
        #print(f"Step {i}: {arr}")

    return arr

print("삽입 정렬 구현을 확인합니다.")
arr = [5, 1, 3, 7, 2, 9]
```

```
print("원래 배열:", arr)
sorted_arr = insertion_sort(arr)
print("정렬된 배열:", sorted_arr)

import random
import time

#1 번
arr1 = [random.randint(1, 100000) for _ in range(100000)]
start_time = time.time()
insertion_sort(arr1)
end_time = time.time()
print(f"100,000 개의 임의 값 정렬 시간: {(end_time - start_time)/60} 분")

#2 번
arr2 = list(range(1, 100001))
start_time = time.time()
insertion_sort(arr2)
end_time = time.time()
print(f"1 부터 100,000 까지의 값 정렬 시간: {end_time - start_time} 초")
```

4. Code 설명

```
def insertion_sort(arr):

    n = len(arr)

    #정렬을 수행할 인덱스 설정, 처음은 1
    for i in range(1, n):
        # 현재 원소를 임시로 저장
        temp = arr[i]
        # 현재 원소의 이전 위치를 가리키는 인덱스를 설정
        j = i - 1

        # 현재 원소를 정렬된 위치에 맞게 삽입
        while j >= 0 and arr[j] > temp:
            # 현재 원소보다 큰 값을 오른쪽으로 한 칸 이동
            arr[j + 1] = arr[j]
            # 인덱스 j를 하나 감소시켜 앞쪽의 원소와 비교
            j -= 1

        # 현재 원소를 정렬된 위치에 삽입
        arr[j + 1] = temp

        # 정렬 과정을 출력합니다.
        #print(f"Step {i}: {arr}")
```

```
    return arr

print("삽입 정렬 구현을 확인합니다.")
arr = [5, 1, 3, 7, 2, 9]
print("원래 배열:", arr)
sorted_arr = insertion_sort(arr)
print("정렬된 배열:", sorted_arr)

import random
import time

#1 번
# 임의의 100,000 개의 값을 지니는 리스트 생성
arr1 = [random.randint(1, 100000) for _ in range(100000)]

start_time = time.time()
insertion_sort(arr1)
end_time = time.time()

print(f"100,000 개의 임의 값 정렬 시간: {(end_time - start_time)/60} 분")

#2 번
# 1 부터 100,000 까지의 값을 지니는 리스트 생성
arr2 = list(range(1, 100001))

start_time = time.time()
insertion_sort(arr2)
end_time = time.time()

print(f"1 부터 100,000 까지의 값 정렬 시간: {end_time - start_time} 초")
```

5. 결과

```
삽입 정렬 구현을 확인합니다.
원래 배열: [5, 1, 3, 7, 2, 9]
Step 1: [1, 5, 3, 7, 2, 9]
Step 2: [1, 3, 5, 7, 2, 9]
Step 3: [1, 3, 5, 7, 2, 9]
Step 4: [1, 2, 3, 5, 7, 9]
Step 5: [1, 2, 3, 5, 7, 9]
정렬된 배열: [1, 2, 3, 5, 7, 9]100,000개의 임의 값 정렬 시간: 28.313254078229267 분
1부터 100,000까지의 값 정렬 시간: 0.07643938064575195 초
```

6. 결과 화면

```
삽입 정렬 구현을 확인합니다.
원래 배열: [5, 1, 3, 7, 2, 9]
Step 1: [1, 5, 3, 7, 2, 9]
Step 2: [1, 3, 5, 7, 2, 9]
Step 3: [1, 3, 5, 7, 2, 9]
Step 4: [1, 2, 3, 5, 7, 9]
Step 5: [1, 2, 3, 5, 7, 9]
정렬된 배열: [1, 2, 3, 5, 7, 9]
```

```
삽입 정렬 구현을 확인합니다.
원래 배열: [5, 1, 3, 7, 2, 9]
정렬된 배열: [1, 2, 3, 5, 7, 9]
100,000개의 임의 값 정렬 시간: 28.313254078229267 분
1부터 100,000까지의 값 정렬 시간: 0.07643938064575195 초
```