알고리즘(3186) 1차 과제

2020.09.20 컴퓨터공학과 201711335 이호용

1. 정렬이 완료된 버블 알고리즘은 더 이상 비교를 수행하지 않고 종료가 가능하게끔

Bool형 변수로 확인하는 방법이 있다.

**\*\* for pass =1 to n-1**

**check= false**

**for i=1 to n-pass**

**if( A[i-1] > A[i])**

**A[i-1] <-> A[i]**

**check=true**

**if(check=false) break**

1. 균등 분포를 가정하고 삽입 정렬을 수행한다.

임의의 i 번째 원소가 선택되어 삽입 정렬이 진행된다고 가정하면 i 번째 원소는 배열 인덱스 상의 0 부터 i 까지 (자리이동 없는 경우도 포함) 자리이동이 진행 될 수 있고 그 확률이 자리 마다 로 같다. 따라서 자리이동 될 인덱스의 기댓값은 각 인덱스\*확률 의 총합이다. 이를 수식으로 나타내고 계산하면 = \* = 이다.

따라서 임의로 선택된 i 번째 원소는 평균적으로 번째 원소와 자리이동 하게 된다.

* 삽입 정렬의 worst case에 의하면 선택된 i 원소는 항상 배열의 맨 앞 (인덱스 0) 으로 위치하는 경우로 매 선택마다 i 번 크기비교가 실행되어야 한다. 하지만 균등 분포에서는 평균적으로 i 번째 원소는 번째에 위치하므로 번 크기비교가 수행되어야 하고 이는 worst case의 시간 복잡도의 절반인 정도로 계산된다.

1. **left -> right 방향으로 가는 변수를 i , right -> left 방향은 변수 j라고 생각.**

**i, j=0**

**While( left+i < right – j )** 왼쪽에서 시작한 탐색 인덱스와 오른쪽에서 시작한 것이 만나면 중지

**if( A[left] < A[left+i] )** 왼쪽 탐색과정에서 피봇보다 큰값이 발견될때

**if( A[left] > A[right-j] )** 오른쪽 탐색을 실행하여 피봇보다 작은 값을 탐색

**A[left+i] <-> A[right-j]** 각 방향 탐색 값끼리 자리이동 실행

**i++ , j++**

**else j++** 오른쪽에서 피봇보다 작은 값이 없으면 j값을 증가시켜(인덱스 감소) 탐색

**else i++** 왼쪽에서 피봇보다 큰값이 없으면 i값을 증가시켜(인덱스 증가) 탐색

**if( left+ i = right- j )** while문 종료후 비교횟수나 원소의 개수에 의해 두 인덱스 값이 같을수도 다를수도 있음

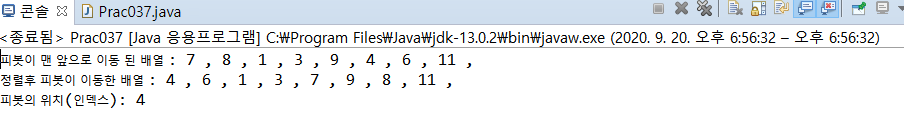
**j <- i**

**return j**

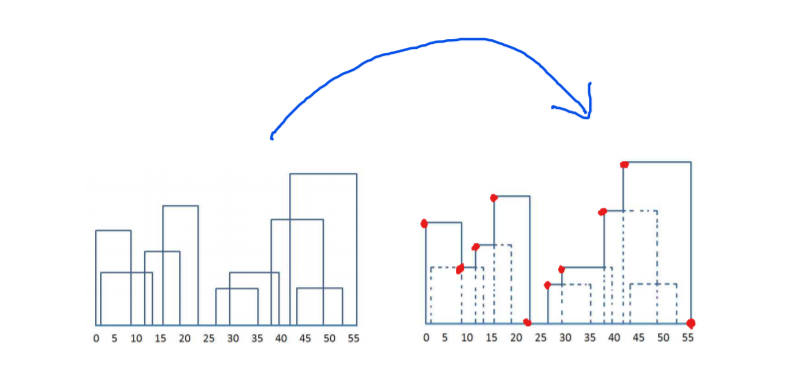
**else**

**j <- i-1**

**return j**

****

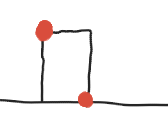
1. skyline ( 공제선 합병 )



Skyline을 그리기 위해서 표시한 빨간 점들이 알고리즘의 결과로 나와야 한다.

우선 입력값 들을 쪼개서 x좌표와 빌딩의 높이를 비교하고 다시 합병해야 한다.

**\*\* ① 기본적으로 하나의 빌딩이라면**

** 이러한 형태의 skyline을 얻을 수 있음**

Input : (leftX , y , rightX) 형태의 배열 building , 배열의 첫 인덱스 a , 끝 인덱스 b

Output : (x, y) 의 형태를 원소로 갖는 배열 skyline

**findSkyline(building , a , b) {**

**if( a = b )** 첫과 끝 인덱스 같음은 원소의 개수가 하나를 말함

**( building의 leftX , y ) append skyline**

**( building의 rightX , 0 ) append skyline**

**return skyline**

**Div <- ( a + b ) / 2**

**A <- findSkyline(building , a , Div)**

**B <- findSkyline(building , Div+1 , b)**

\*\*A 와 B에는 (x ,y)형태의 배열이 저장되어 merge로 넘어감

**return merge(A, B)**

\*\* merge된 skyline 리턴

**}**

**merge(A, B) {**

**i=0, j=0 ,h1=0 , h2=0**

**while( i < A의 개수 and j < B의 개수 ){**

**if ( A[i]의 leftX < B[j]의 leftX )**

**x1 = A[i]의 leftX**

**h1 = A[i]의 y**

**i++**

**\*\*② Max(h1,h2)의 값이 skyline의 마지막 원소의 y값과 같다면 skyline에 새로운 값을 추가하지 않음**

**\*\* skyline의 마지막 원소의 x값과 x1값이 같다면 skyline에는 (x1, Max(skyline의 마지막원소의 y값 , h1))을 추가함**

**\*\* 위의 조건에 해당하지 않는다면 skyline에 (x1, Max(h1,h2)) 추가**

**Else**

**x2 = B[j] 의 leftX**

**h2 = B[j] 의 y**

**j++**

**}**

**\*\* i 나 j 의 값이 A 혹은 B의 개수 까지 도달하지 못했을때는 해당 배열의 나머지 원소들을 skyline에 추가**

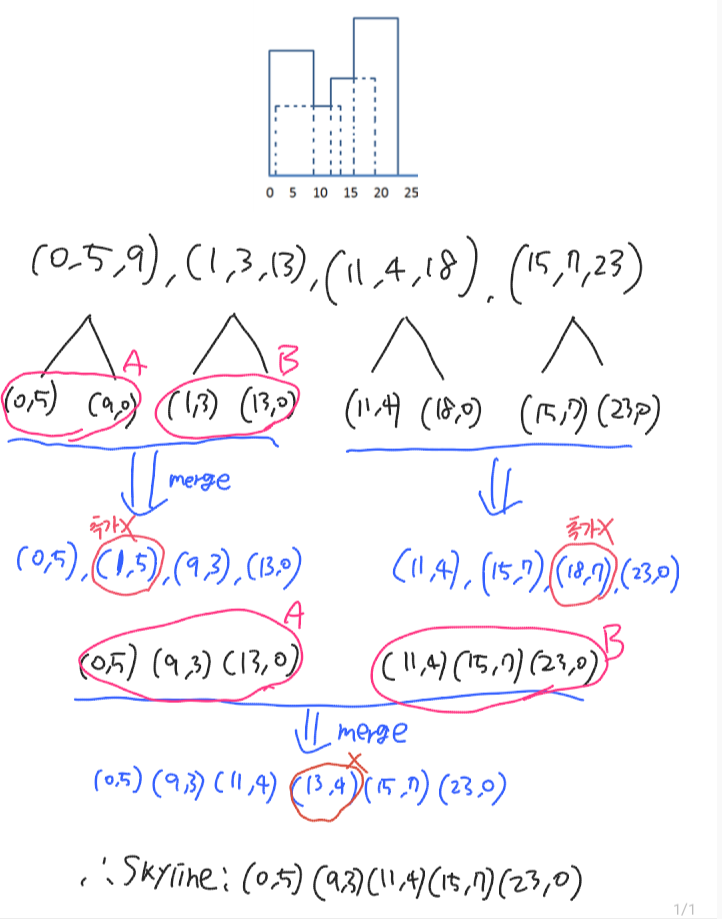
**return skyline**

**}**

시간 복잡도는 한번 분할이 실행될 때 문제의 크기가 1/2이 되고 문제의 개수가 2가 되고 각 단계에서 이 소요 된다

+ 이므로 시간 복잡도는 이 된다.

한글로 설명한 부분의 이해를 돕기 위해 예시를 추가하였습니다.

****