3. java

import java.util.Arrays;  
import java.util.Scanner;  
  
public class GasStation {  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
 int n = sc.nextInt(); // 도시의 개수  
 int[] roadLen = new int[n-1];  
 int[] gasVal = new int[n];  
 int totalLen = 0;  
 int cost = 0;  
 int driven = 0;  
 // 도로의 길이와 리터당 가격을 저장  
 for (int i = 0; i < n-1; i++) {  
 roadLen[i] = sc.nextInt();  
 totalLen += roadLen[i]; // 전체 거리를 계산  
 }  
 for (int i = 0; i < n; i++) {  
 gasVal[i] = sc.nextInt();  
 }  
 int minGas = Arrays.*stream*(gasVal).min().getAsInt(); // 기름값이 가장 낮은 곳을 저장한다  
 for (int i = 0; i < gasVal.length; i++) {  
 if (gasVal[i] == minGas) {  
 // 만약 현재 위치가 경로상 가상 싼 주유소라면  
 cost += totalLen\*gasVal[i]; // 남은거리\*리터당가격  
 break;  
 }  
 else {  
 // 아니라면 현재 주유소 보다 싼 주유소가 나올때 까지 채운다  
 for (int j = i+1; j < gasVal.length; j++) {  
 if (gasVal[i]>gasVal[j]) {  
 // 다음 주유소 보다 비싸다면 j 번째 전까지 충전한다  
 driven += roadLen[j-1]; // j번째 주유소 까지 이동하고  
 totalLen -= driven; // 총 거리에서 주행거리를 빼준다  
 cost += driven\*gasVal[i]; // 현재 주유소\*주행거리를 비용에 더해준다  
 driven=0; // 주행거리를 다시 0으로 초기화  
 break;  
 }  
 else {  
 // 주행거리를 즐려준다  
 driven += roadLen[j-1];  
 }  
 }  
 if(totalLen==0){  
 // 거리를 모두 이동했다면 반복을 멈춘다  
 break;  
 }  
 }  
 }  
 System.*out*.println(cost);  
 }  
}

리뷰: 경로상의 각 주유소를 확인하여 현재 주유소를 기준으로 이후 더 값싼 주유소가 등장 할때까지 이동한뒤 움직인 거리가 총 거리에 다다를때 까지 반복을 하며 금액을 증가시켜주었습니다. for문을 두번 사용하여 복잡도가 O(n^2)이 되었는데 이를 개선할 방법이 있을지 더 최적화된 방법을 좀더 연구해봐야 될것같습니다.

3.python

n = int(input()) # 도시의 개수  
dist = list(map(int, input().split())) # 주유소간 거리  
price = list(map(int, input().split())) # 주유소의 가격  
cost = 0 # 비용을 0으로 초기화  
totalDist = sum(dist) # 도시간 총거리  
for i in range(n):  
 if price[i] == min(price): # 현재 주유소가 최저값 주유소라면  
 cost += totalDist\*price[i] # 가격은 남은거리\*현재주유소가격  
 break # 더이상 반복이 필요 없으므로 종료  
 else: # 만약 최저값이 아니라면  
 driven = 0 # 현재 주유소에서 다음 주유소까지의 주행거리  
 for j in range(i+1,n):  
 if price[i] > price[j]: # 만약 현재 주유소가 다음 주유소보다 비싸다면  
 driven = sum(dist[i:j]) # i번째 주유소에서 j번째 주유소까지의 거리  
 totalDist -= driven # 총거리에서 주행거리를 빼준다  
 cost += driven\*price[i] # j번째 주유소까지의 비용  
 break  
 if totalDist == 0: break # 총거리를 모두 주행했다면 반복 종료  
print(cost)

리뷰: 만약 현재 주유소가 최저값 주유소가 아니라면 다음 주유소에 무조건 현재 주유소보다 싼 주유소가 있을것 이므로 만약 최저값 주유소가 아니라면 현재보다 싼 주유소가 나올때까지 반복하며 거리에 따른 비용을 더해준다 그리고 주행거리가 총거리와 동일해 졌을때 반복을 종료하여 비용을 받아왔다

4. java

import java.util.Arrays;  
import java.util.Comparator;  
import java.util.Scanner;  
  
public class VectorSort {  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
 int n = sc.nextInt(); // 좌표의 개수  
 int[][] vector = new int[n][2];  
 for (int i = 0; i < n; i++) {  
 vector[i][0] = sc.nextInt();  
 vector[i][1] = sc.nextInt();  
 }  
 Arrays.*sort*(vector, new Comparator<int[]>() {  
 @Override  
 public int compare(int[] o1, int[] o2) {  
 if (o1[0]==o2[0]){  
 // 만약 x좌표가 같다면  
 return o1[1] - o2[1]; // y좌표를 내림차순으로  
 }else{ // 아니라면 x좌표를 오름차순으로  
 return o1[0] - o2[0];  
 }  
 }  
 });  
 for (int[] xyVector : vector) {  
 for (int xy : xyVector) {  
 System.*out*.print(xy+" ");  
 }  
 System.*out*.println();  
 }  
 }  
}

리뷰: 이번에는 두 인자값을 비교하여 같을시 다시 정렬 조건을 설정하는 문제가 핵심이었습니다. 객체간의 비교를 할때에는 theComparing으로 첫번째 비교이후의 비교값을 설정하였다면 이번에는 Comparator를 직접 설정하여 만약 값이 같을때는 y좌표를 기준으로 정렬이 되도록 해보았습니다. lambda 식을 통해 하는 방법을 찾았으나 식 하나로 완성하는 법을 발견하지 못하여 compare 메서드를 오버라이드하여 직접 정렬 조건을 만들었습니다. 앞으로도 자바 실력의 부족함을 느꼈습니다.

4.python

n = int(input()) # 점의 개수  
vector = [list(map(int, input().split())) for \_ in range(n)] # n 개의 점의 좌표를 받아온다  
vector.sort(key= lambda x:(x[0],x[1])) # x좌표를 기준으로 정렬 같다면 y를 기준으로 오름차순  
for x, y in vector:  
 print(x,y) # 출력 형식에 맞게 출력해준다

리뷰: 파이썬에서 sort 의 key옵션을 통하여 0번째 정렬조건인 x좌표와 다음 정렬 조건 y를 설정하여 손쉽게 정렬할 수 있었습니다.