Algorithm Assignment #4

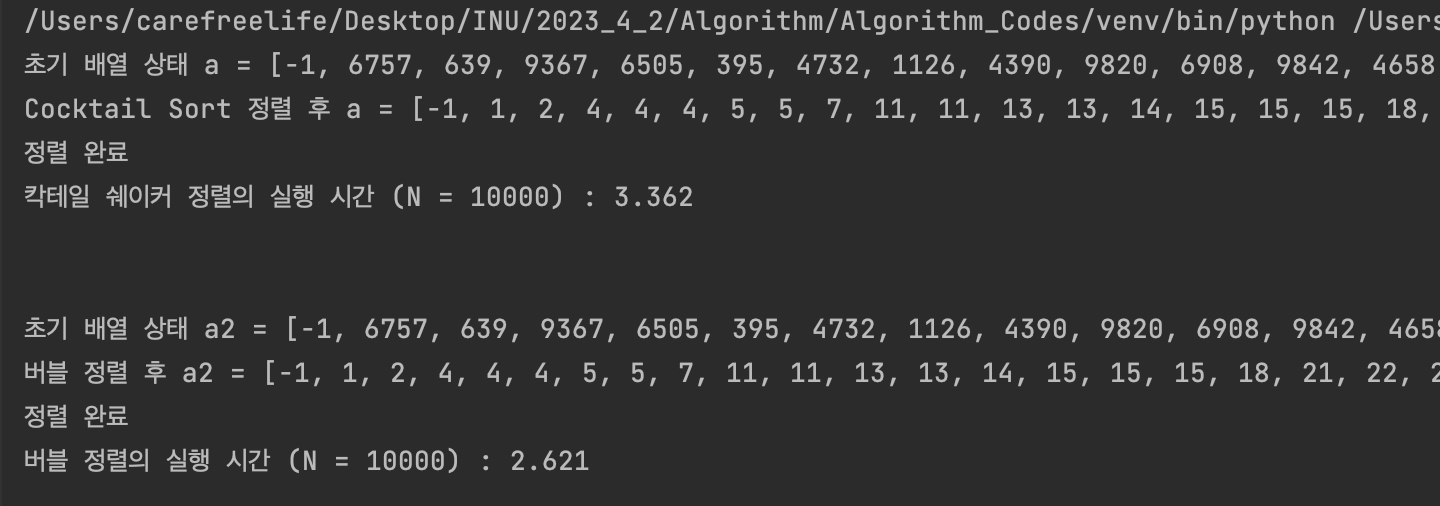
채진석 교수님

201702797 채승민

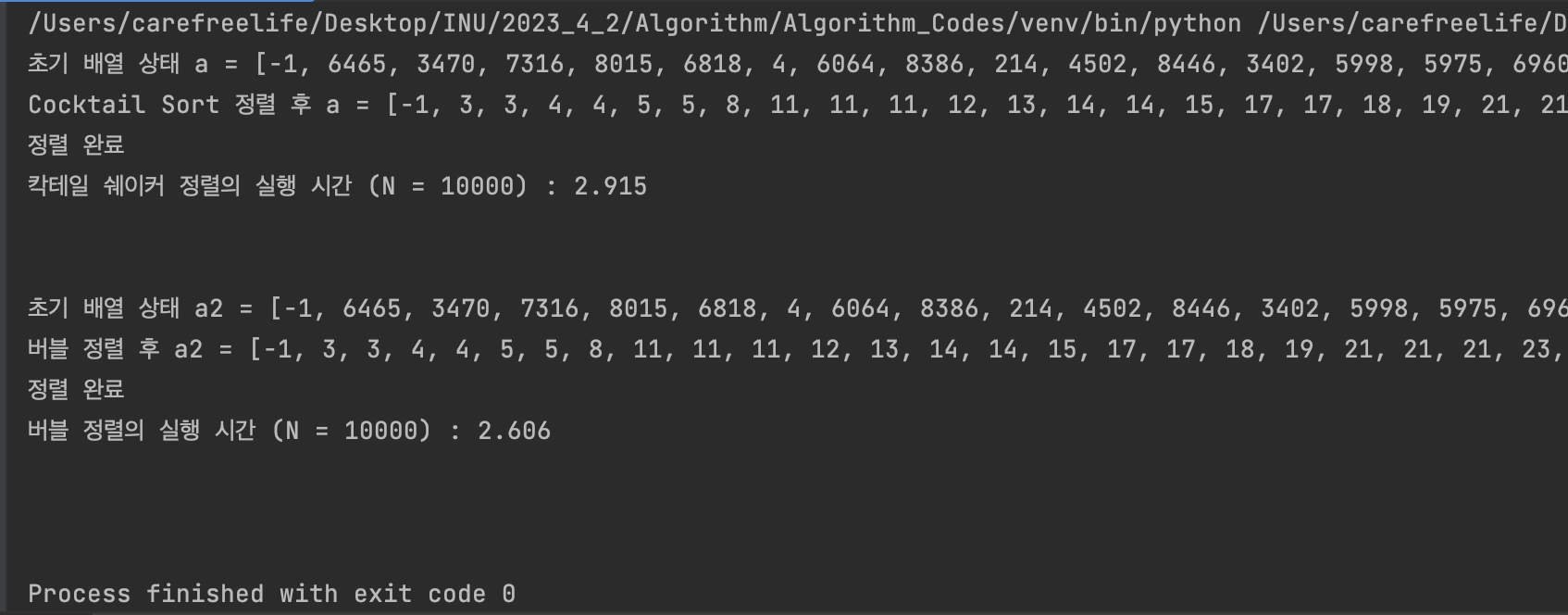
[Cocktail Shaker Sort - ADL]

|  |
| --- |
| algorithm cocktailShakerSort(a[], n)  toggle <- false;  for (i <- 1 to n – 1) do {  toggle <- not toggle;  if toggle then {  for (j <- i - i // 2 to n) do {  if (a[j] > a[j + 1]) then {  swap(a[j], a[j + 1]);  }  }  }  Else {  for (j <- n – i // 2; j <- 0; j <- j - 1) do {  if a[j] < a[j - 1] then {  swap(a[j], a[j - 1]);  }  }  }  }  end; |

[Bubble Sort 와 비교]



* 칵테일 쉐이커 정렬이 조금 더 느린 것으로 보인다.

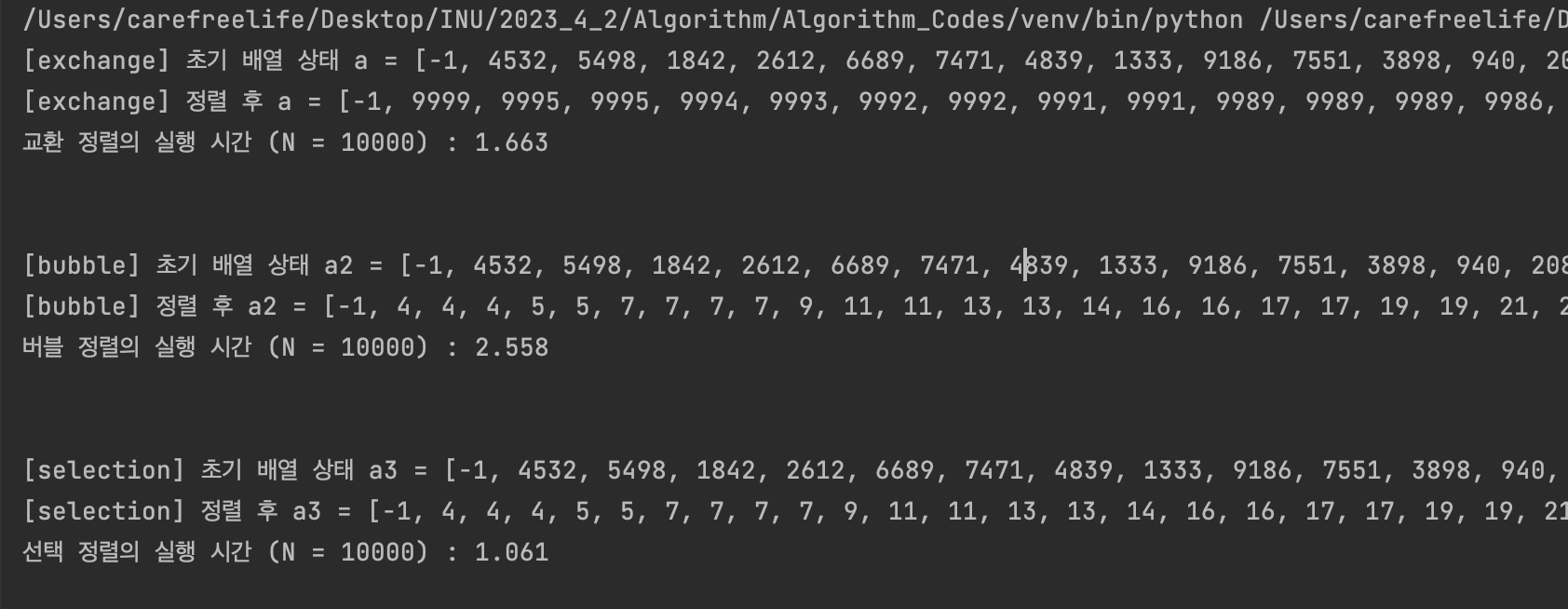


* 교수님께서 올려주신 코드로 실행한 경우에도 버블정렬이 더 빠른 것으로 나타난다.

[Exchange Sort - ADL]

|  |
| --- |
| algorithm exchange\_sort(a, n)  for (i <- 1; i < n +1; i <- i +1) do {  for j <- i; j < n + 1; j <- j +1) do {  if (a[i] < a[j]) then {  swap (a[i], a[j]);  }  }  }  end; |

[Bubble / Selection Sort 와 비교]



* Selection – Exchange – Bubble 순서로 정렬이 빠르다.
* Exchange sort 를 역순으로 재정렬 하거나 나머지 정렬 알고리즘을 역순으로 만드는 과정에서 성능이 바뀔 수 있다.

[Natural Merge Sort - ADL]

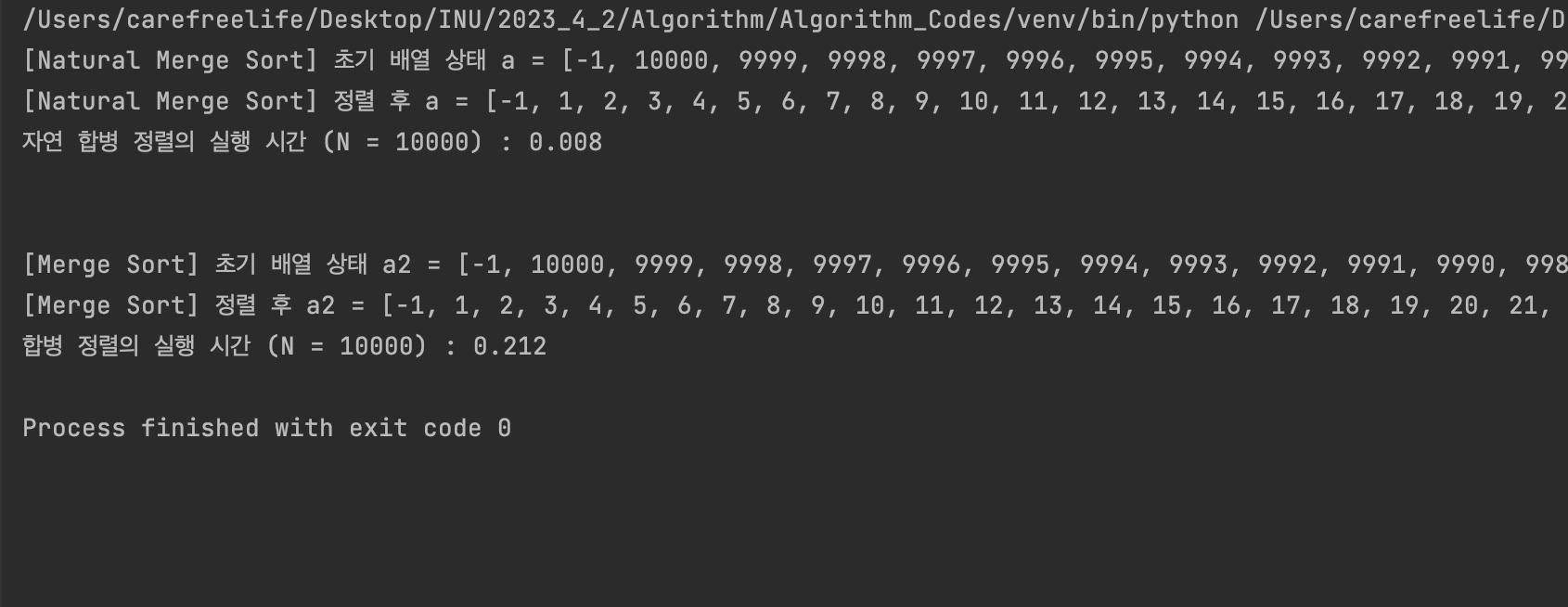
|  |
| --- |
| algorithm mergeFromSorted(l1[], l2[]) {  i1 <- 0;  i2 <- 0;  len\_l1 <- length(l1);  len\_l2 <- length(l2);  merged <- [];  while (i1 < len\_l1 OR i2 < len\_l2) do  {  if (i1 >= len\_l1 OR i2 >= len\_l2) then  {  merged <- merged + l1[i1:] + l2[i2:];  break;  }  if (l1[i1] <= l2[i2]) then  {  Merged에 l1[i1] 삽입;  i1 <- i1 + 1;  }  else  {  Merged에 l2[i2] 삽입;  i2 <- i2 + 1;  }  }  return merged;  }  END;  algorithm generateRuns(l[])  {  if (length(l) <= 1) then  {  return l;  }  len\_l <- length(l);  prev\_item <- l[0];  runs <- [];  run <- [prev\_item];  i <- 1;  while (i < len\_l) do  {  if (prev\_item > l[i]) then  {  runs 에 run삽입;  run <- [l[i]];  }  else  {  Run에 l[i] 삽입;  }  prev\_item <- l[i];  i <- i + 1;  }  Runs에 run 삽입; // add last run  return runs;  }  END;  algorithm naturalMergeSort(runs[])  {  len\_runs <- length(runs);  if (len\_runs <= 1) then  {  return runs;  }  while (len\_runs > 1) do  {  rst <- [];  for (i <- 0; i <= len\_runs – 1; i = i +2) do  {  if (i == len\_runs – 1) then  {  rst에 runs[i] 삽입;  }  else  {  rst에 mergeFromSorted(runs[i], runs[i + 1] 삽입);  }  }  runs <- rst;  len\_runs <- length(runs);  }  return rst[0];  }  END; |

[Merge Sort 와 비교]

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

* Merge Sort 에 비해 월등히 좋은 성능을 보여준다.



* 역순으로 정렬되어 있는 데이터도 Natural Merge Sort 가 더 좋은 성능을 보여준다..

[Coding Test #2 – Source Code & Result]

import random  
  
import sys, math  
# input = sys.stdin.readline  
  
def remain\_combination(a, n, m):  
 # 나머지만 담은 배열.  
 rem = [0] \* m  
 rem[a[0] % m] += 1  
  
 for i in range(1, n):  
 a[i] = (a[i-1] + a[i]) % m  
 rem[a[i]] += 1  
  
 # 0의 개수 + 나머지가 같은 것들의 조합  
 ans = rem[0]  
 for i in rem:  
 if i >= 2:  
 ans += math.comb(i, 2)  
 print(ans)  
  
  
def get\_input():  
 a = []  
 N = int(input("수의 개수 N 입력: "))  
 M = int(input("나누어 떨어질 수 M 입력: "))  
  
 for i in range(N + 1):  
 a.append(random.randint(0, 1000000000))  
 return a, N, M  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 a, N, M = get\_input()  
 # n, m = map(int, input().split())  
 # a = list(map(int, input().split()))  
 print(f"배열 = {a}\n N = {N}, M = {M}")  
 remain\_combination(a, N, M)

[Results]

텍스트, 스크린샷, 폰트, 블랙이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명