Homework #6
(Introduction to Data Structures)

Due date: June 17, 2018

학번: 2014310407

이름: 이 준 혁(JunHyuk Lee)

1. Calculating the Sizes of the Islands

1.1. Solution

모든 노드를 확인해야 하는데, 어떤 특정 위치를 방문했는지를 확인했는 지를 확인하기 위해 mark 라는 2 차원 배열을 선언했다. 이 때 어떤 위치에 섬이 없다면, 방문할 필요가 없으므로, 미리 방문했다는 의미에서 1 로 채우고, 섬이 있는 경우에는 0 으로 채웠다. 그리고 노드를 방문할 때 마다 mark 값을 1 로 바꿨다.

그리고 만약 i, j 번째 위치의 mark 값이 0 이라면, 즉 방문 된 적이 없다면, 해당 위치의 mark 값을 1 로 바꾼 후, 그 위치를 저장했다. 이후 그 노드와 연결된 모든 노드들을 다 방문해야 하는데, 이 과정에서 가장 효율적으로 모든 연결된 노드들을 찾는 방법은 BFS 를 이용하는 것이라 생각했다. 따라서 queue 를 이용했다. 큐를 이용해 BFS 와 같은 방법으로, 노드가 발견될 때 마다 큐에 넣고, 큐에서 원소를 꺼낸 후에는 상하좌우를 살펴보며 또 방문 되지 않은 노드가 있는지 살펴보았다. 만약 방문 되지 않은 노드가 있다면, 방문하면서 큐에 넣고, mark 값을 1 로 바꾸었다.

이 과정을 queue 가 빌 때까지 했다. 이렇게 한 번의 과정이 끝나면, 한 섬에 대해 search 를 모두 다 시행한 것이 된다. 따라서 섬 개수를 한 개 증가시키고, 해당 섬에 대해 총 방문한 노드 수를 저장했다.

이 과정을 모든 i, j 에 대해 시행한다면 총 섬의 개수와 섬의 크기에 대한 정보를 얻을 것이다.
이를 배열로 저장한 후에, 내장 함수를 이용해 sorting 했다. 이후 이 정보를 반환했다.

1.2. Result (snapshot)

2.

2.1. Solution

2.2. Result (snapshot)

3. Melting Cheese

3.1. Solution

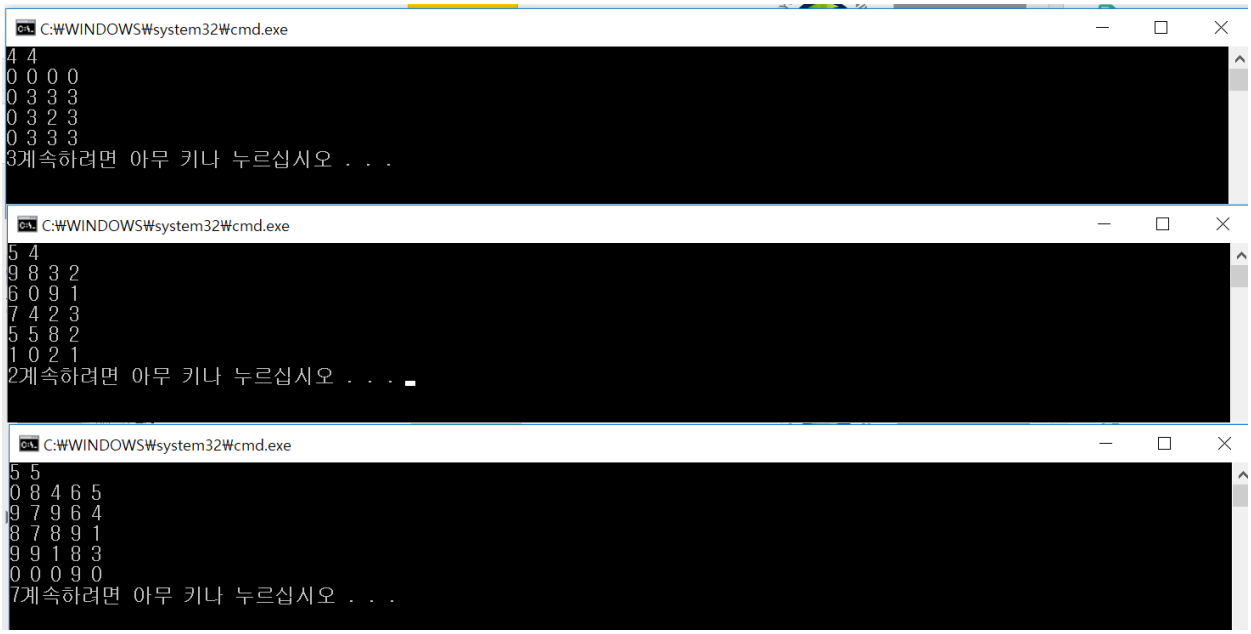
우선 치즈가 연결되어 있는지 판단하는 함수가 필요하다고 생각해서, 이차원 배열(치즈)을 입력으로 받아서 연결 여부를 판단하는 함수를 구성했다.

이를 위해 1 번 문제의 아이디어를 이용해, 치즈의 개수를 구한 후, 치즈의 개수가 1 개가 아닌 경우 false 를, 개수가 1 인 경우 치즈가 모두 연결되어 있는 것이므로 true 를 반환했다.

이를 위해 1 번 문제와 같이 큐를 이용해, BFS 를 시행하며 치즈의 개수를 구했다.

따라서 원래 함수로 돌아와서, 치즈가 연결되어 있으면, 시간을 1 증가시킨 후, 치즈를 나타내는 값이 0 이 아닌 경우, 1 을 감소시켰다. 이 과정을 치즈가 연결되어 있지 않을 때까지 반복한 후, 최종 결과로 나온 시간을 반환했다. 이는 치즈가 끊어질 때까지 걸리는 시간이 된다.

3.2. Result (snapshot)



4. Two Degrees of Separation

4.1. Solution

다음과 같은 두 가지 단계로 나누어서 진행하려 했다.

1. Alice, Bob 의 친구들을 표시

2. 1 의 정보를 이용해 Alice, Bob 의 친구의 친구들을 표시

3. Alice, Bob 의 친구들과 친구의 친구들의 수를 센 후 반환

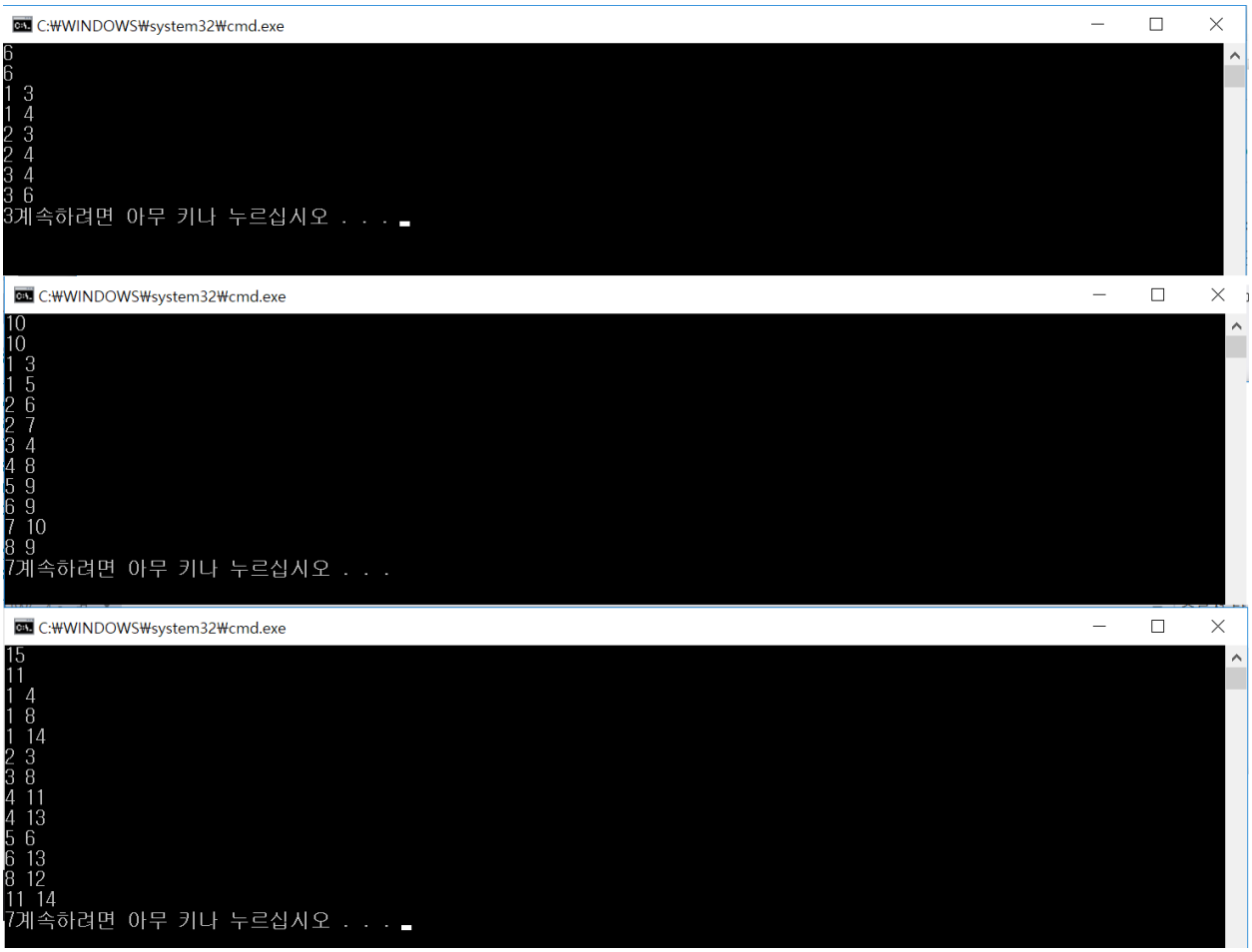
우선 i 번째 사람이 Alice, Bob 의 친구인지를 나타내는 배열 friends 를 선언했다. 이 값이 1 이면 Alice, Bob 의 친구 혹은 친구의 친구인 것이다. 우선 friends 의 값을 다 0 으로 초기화했다.

이를 채우기 위해 우선 1 단계인, Alice, Bob 의 친구들을 다 표시했다. Fillfriends 라는 함수를 이용해, 만약 i 번째 사람이 Alice, Bob 과의 친구라면, friends[i]를 -1 로 초기화했다. 이를 Alice 에 대해 한 번, Bob 에 대해 한 번 시행했다. 이후 -1 값을 다 1 로 바꾸어 주었다.

다시, 2 단계를 위해 friends[i]가 1 인 i 들에 대해, Fillfriends 함수를 다시 사용해 i 의 친구들을-1 로 초기화 했다. 이 때 1 이 아닌 -1 로 초기화 한 이유는, 1 단계에서 Alice, Bob 의 친구들에 대해서만 시행해야 하는데, -1 이 아닌 1 로 초기화했다면, 이 과정이 꼬이게 된다. 즉, Alice, Bob 의 친구의 친구들에 대해서도 친구를 채우는 행위를 시행하게 된다. 이런 잘못된 반복을 없애고, 2 번째 친구와 1 번째 친구를 구별하기 위해 -1 로 초기화했다. 이후 -1 인 값들을 1 로 다 바꾸어 주었다.

이후 friend 에 저장된 모든 값을 다 더함으로써 모든 친구 및 친구의 친구들의 수를 세었다. 이 값을 반환했다.

4.2. Result (snapshot)



5.

5.1. Solution

5.2. Result (snapshot)

6. Comparing Sorting Algorithms

6.1. Solution

1. Randomzied data set

Algorithm	Running time(sec)	# of comparisons
Bubble sort	31.272	4999950000
Selection sort	13.051	4999950000
Insertion sort	7.234	2501588689
Shell sort	0.029	3956410
Merge sort	0.088	1668928
Iterative Merge sort	0.081	1700000
Heap sort	0.053	3249500
Randomized Quick sort	0.024	1953337
Quick sort	0.019	1957316

1) Bubble sort

비교 개수가 정확히 n*(n+1)/2 로 나오는 것을 확인할 수 있었다. 매 비교마다 위치를 바꿈으로 인해 가장 시간이 오래 걸리는 것을 확인했다.

2) Selection sort

역시 비교 개수가 정확히 n*(n+1)/2 로 나오는 것을 확인할 수 있었다. 최솟값을 찾은 후 위치를 바꿨기 때문에, Bubble sort 에 비해 걸리는 시간은 비교적 적은 것을 확인할 수 있었다.

3) Insertion sort

역시 O(n^2)의 time complexity 를 보였으나, bubble 이나 selection 보다 적은 비교 횟수 및 시간 내에 정렬이 끝내는 것을 확인할 수 있었다.

4) Shell sort

비교 횟수는 다른 nlogn 알고리즘보다 다소 많으나, recursion 을 사용하지 않아서 시간 측면에서 굉장히 빠른 것을 확인할 수 있었다.

5) Merge sort

비교 횟수는 가장 적었다. 하지만 많은 recursion 으로 인해 실제 구현에서 시간이 다소 더 소요되는 결과를 얻었다.

6) Iterative Merge sort

iterative 하게 구성할 경우 마지막 부분에서 잘리기 때문에 비교 횟수는 조금 더 많은 것을 확인했다. 다만 recursion 이 없으므로 시간은 덜 소요되는 것을 확인했다.

7) Heap sort

비교횟수는 nlogn 알고리즘 중 가장 많았으나, recursion 을 적게 사용해서 시간 측면에서 빠른 결과를 얻을 수 있었다.

8) Randomized Quick sort

Quick sort 보다는 조금 느렸지만 굉장히 빠른 시간과 적은 비교횟수로 비교를 끝마쳤다. 시간이 Quick sort 보다 더 걸린 이유는 처음에 random 하게 pivot 을 잡고 그 원소를 첫 번째 원소와 바꾸는 시간이 존재해서 그런 것으로 보인다.

9) Quick sort

recursion 이 있음에도 불구하고 가장 빠른 시간 안에 정렬을 완료했다.

2. Sorted data set

Algorithm	**Running time(sec)**	**# of comparisons**
Bubble sort	20.447	4999950000
Selection sort	12.326	4999950000
Insertion sort	15.643	5000049928
Shell sort	0.013	2218993
Merge sort	0.083	1668928
Iterative Merge sort	0.073	1700000
Heap sort	0.048	3094868
Randomized Quick sort	0.022	2182497
Quick sort	Stack overflow	Stack overflow

1) Bubble sort

randomized 되어 있을 때와 비교 횟수는 같은 것을 확인했다. 다만 시간은 덜 걸렸는데, 이 부분은 설명할 수 없다. 역순으로 정렬된 데이터를 bubble sort 를 하면 매 순간마다 원소의 위치를 바꿔야 한다. 그래서 시간이 더 걸릴 것이라 예측했으나 오히려 더 시간이 줄어들었다.

2) Selection sort

역시 비교 횟수는 randomized data set 일 때와 같은 횟수를 가졌다. 어차피 selection sort 는 최솟값을 찾아서 바꾸는 것이므로, data 가 sorted 되어 있는지 여부와 관계없이 같은 시간이 소요되었다.

3) Insertion sort

Insertion sort 의 worst case 이다. 각 데이터마다 최대로 비교해서 맨 앞에 삽입하는데, 비교 횟수가 bubble, selection 보다 한번 더 많으므로, 위의 두 정렬보다 비교 횟수가 더 많았다.

4) Shell sort

Insertion sort 와 마찬가지로 O(n^2)이 나올 줄 알았으나 오히려 비교 횟수가 더 줄었다. 굉장히 빠른 시간 내에 정렬을 완료했다.

5) Merge sort

Data 의 정렬 여부와 비교 횟수의 차이가 없어서, 정확히 같은 비교 횟수로 정렬을 완료했다. 따라서 시간도 비슷한 결과로 나왔다.

6) Iterative Merge sort

Merge sort 와 마찬가지로 같은 비교 횟수로 정렬을 완료했다. 소요 시간도 randomized data set 과 비슷한 결과로 나왔다.

7) Heap sort

Max heap 을 통해 구현했는데, 처음 데이터가 들어올 때 정렬되어 있는 상태, 즉 max heap 의 상태로 입력을 받으니 처음 data 를 heap 으로 만드는 과정에서 소요되는 시간이 줄어들었다. 하지만 그 이후에 데이터를 정렬하면서 heap 을 유지시키는 과정에서 드는 비용은 동일하므로, 조금 시간과 비교 횟수가 줄어들긴 했지만, 비슷한 결과를 얻을 수 있었다.

8) Randomized Quick sort

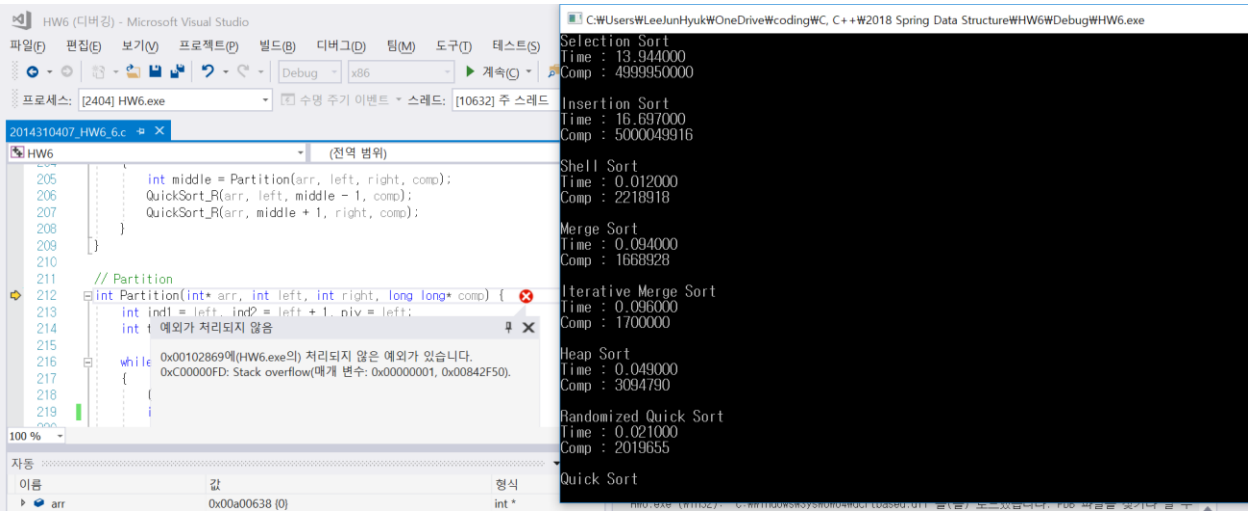
pivot 을 random 하게 잡기 때문에 data 의 정렬 여부와 상관없다. 따라서 randomized data set 일 때와 거의 같은 비교 횟수와 시간 내에 정렬을 완료했다.

9) Quick sort

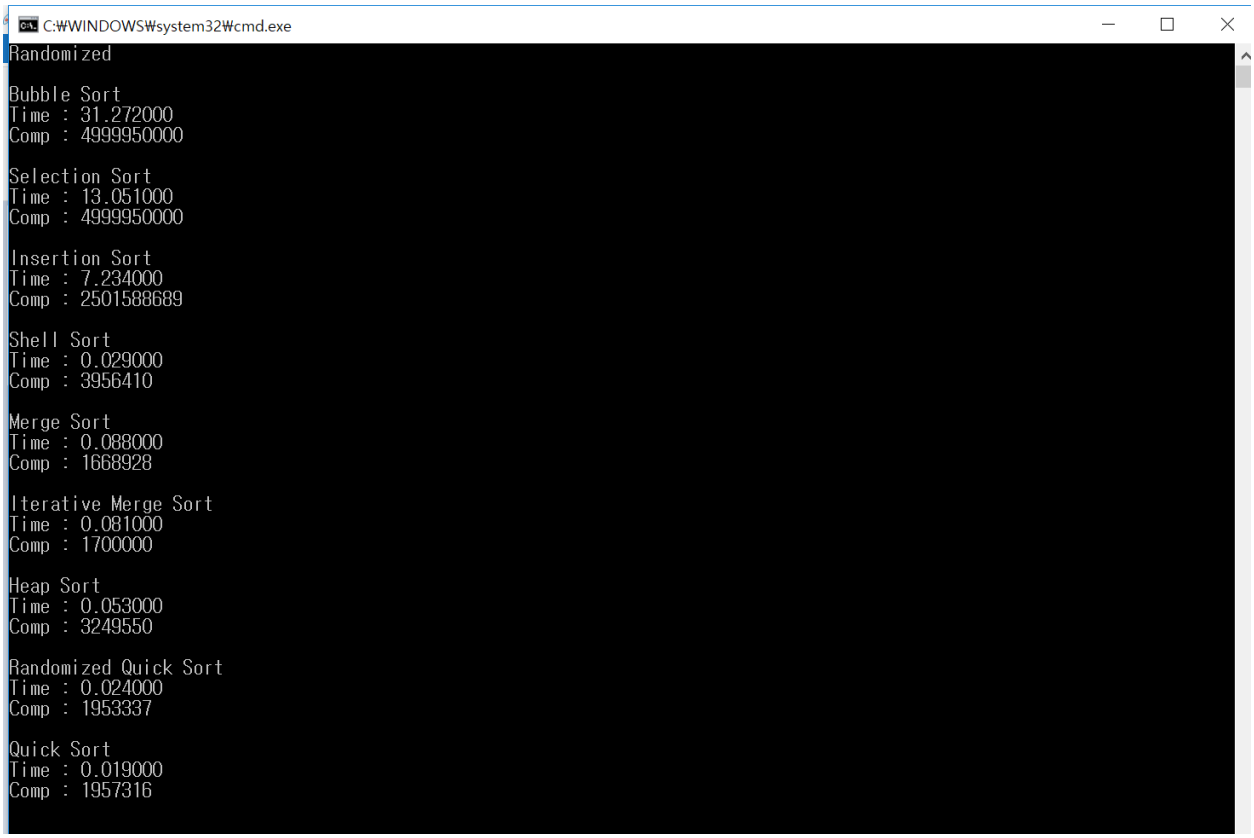
Sorted data set 에서 quick sort 는 정렬하지 못했다. 그 이유는, 다음과 같아 stack overflow 가 발생했기 때문이다. 이 현상이 일어난 이유는, data set 이 order 돼 있는 경우에, quick sort 의 time complexity 는 O(n^2)이고, 함수를 recursive 하게 call 하므로 함수를 최소 O(n)

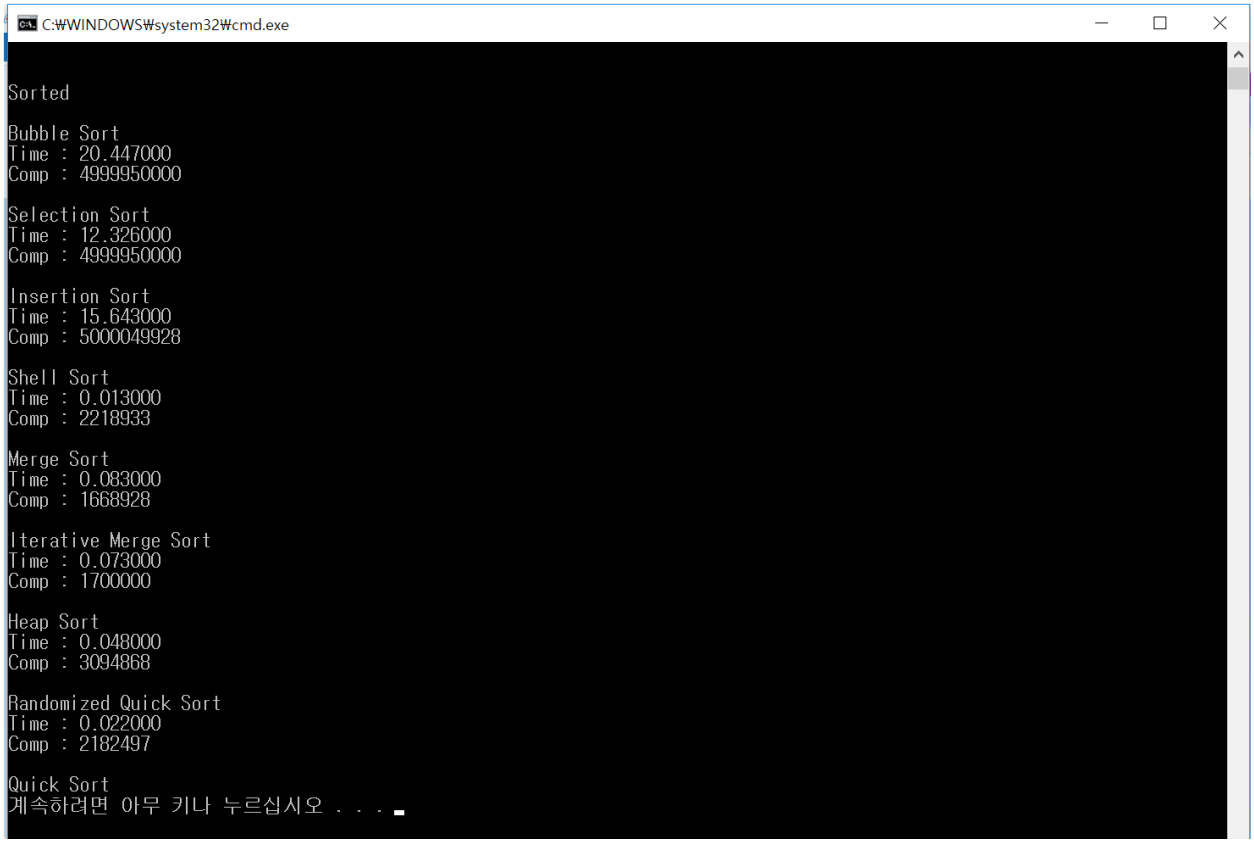
번 호출한다.

따라서 함수를 위한 space complextity 가 O(n)이 필요한데, 이는 메모리에서 스택 영역을 사용한다. 하지만 스택 영역에는 분명히 메모리 제한이 있고, n 이 커지면 이 스택 영역을 초과할 정도의 메모리를 필요로 하기 때문에, stack overflow 가 발생하게 된다. 따라서 original quick sort 로는 이 경우에 sorting 할 수 없었다.



6.2. Result (snapshot)


C:₩WINDOWS₩system32₩cmd.exe
Randomized
Bubble Sort
Time : 31.272000
Comp : 4999950000
Selection Sort
Time : 13.051000
Comp : 4999950000
Insertion Sort
Time : 7.234000
Comp : 2501588689
Shell Sort
Time : 0.029000
Comp : 3956410
Merge Sort
Time : 0.088000
Comp : 1668928
Iterative Merge Sort
Time : 0.081000
Comp : 1700000
Heap Sort
Time : 0.053000
Comp : 3249550
Randomized Quick Sort
Time : 0.024000
Comp : 1953337
Quick Sort
Time : 0.019000
Comp : 1957316


C:₩WINDOWS₩system32₩cmd.exe
Sorted
Bubble Sort
Time : 20.447000
Comp : 4999950000
Selection Sort
Time : 12.326000
Comp : 4999950000
Insertion Sort
Time : 15.643000
Comp : 5000049928
Shell Sort
Time : 0.013000
Comp : 2218933
Merge Sort
Time : 0.083000
Comp : 1668928
Iterative Merge Sort
Time : 0.073000
Comp : 1700000
Heap Sort
Time : 0.048000
Comp : 3094868
Randomized Quick Sort
Time : 0.022000
Comp : 2182497
Quick Sort
계속하려면 아무 키나 누르십시오 . . .