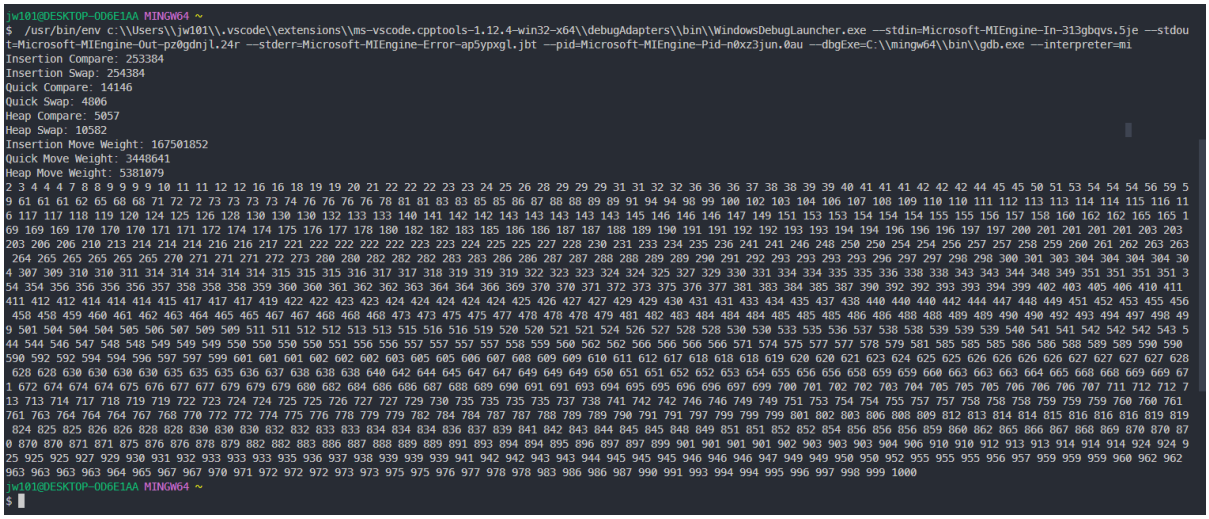
```
#include <stdio.h>
#include <stdlib.h>

#include <algorithm>
#include <iostream>

using namespace std;

const int n = 1000;
int insertionCompare = 0;
int insertionSwap = 0;
int QuickCompare = 0;
int QuickSwap = 0;
int heapCompare = 0;
int heapSwap = 0;

long long insertionMoveWeight = 0;
long long QuickMoveWeight = 0;
long long heapMoveWeight = 0;

void insertion(int a[], int N) {
	int i, j, v;
	for (i = 0; i < N; i++) {
		v = a[i];
		insertionMoveWeight += a[i];
		j = i;
		while (j > 0 && a[j - 1] > v) {
			insertionCompare++;
			a[j] = a[j - 1];
			insertionMoveWeight += a[j - 1];
			insertionSwap++;
			j--;
		}
		a[j] = v;
		insertionMoveWeight += v;
		insertionSwap++;
	}
}
```

```
void swap(int a[], int i, int j) {
	int t = a[i];
	a[i] = a[j];
	a[j] = t;
}

int partition(int a[], int l, int r) {
	int i, j, v;
	if (r > l) {
		v = a[l];
		i = l;
		j = r + 1;
		while (1) {
			do {
				++i;
				QuickCompare++;
			} while (a[i] < v);
			do {
				--j;
				QuickCompare++;
			} while (a[j] > v);
			if (i >= j) break;
			swap(a, i, j);
			QuickSwap += 2;
			QuickMoveWeight += a[i] * 2 + a[j];
		}
		swap(a, j, l);
		QuickSwap += 2;
		QuickMoveWeight += a[j] * 2 + a[l];
	}
	return j;
}

void quicksort(int a[], int l, int r) {
	int j;
	if (r > l) {
		j = partition(a, l, r);
		quicksort(a, l, j - 1);
		quicksort(a, j + 1, r);
```

```
        }
    }

    void makeHeap(int a[], int i, int N) {
        int j, v;
        v = a[i];
        heapMoveWeight += a[i];
        j = 2 * i;
        while (j <= N) {
            if (j < N && a[j] < a[j + 1]) {
                j++;
                heapCompare++;
            }
            if (v >= a[j]) {
                break;
                heapCompare++;
            }
            a[j / 2] = a[j];
            heapSwap++;
            heapMoveWeight += a[j];
            j *= 2;
        }
        a[j / 2] = v;
        heapSwap++;
        heapMoveWeight += v;
    }

    void heapsort(int a[], int n) {
        int i;

        for (int i = n / 2; i > 0; i--) {
            makeHeap(a, i - 1, n - 1);
        }
        for (i = n - 1; i > 0; i--) {
            swap(a, 0, i);
            makeHeap(a, 0, i - 1);
        }
    }
```

```
int main() {
	int a[n], b[n], c[n];
	for (int i = 0; i < n; i++) {
		a[i] = rand() % 1000;
		b[i] = a[i];
		c[i] = a[i];
		// printf("%5d", a[i]);
	}

	insertion(a, n);
	quicksort(b, 0, n - 1);
	heapsort(c, n);

	cout << "Insertion Compare: " << insertionCompare << endl;
	cout << "Insertion Swap: " << insertionSwap << endl;
	cout << "Quick Compare: " << QuickCompare << endl;
	cout << "Quick Swap: " << QuickSwap << endl;
	cout << "Heap Compare: " << heapCompare << endl;
	cout << "Heap Swap: " << heapSwap << endl;

	cout << "Insertion Move Weight: " << insertionMoveWeight << endl;
	cout << "Quick Move Weight: " << QuickMoveWeight << endl;
	cout << "Heap Move Weight: " << heapMoveWeight << endl;

	// print a[i]
	for (int i = 0; i < n; i++) {
		cout << a[i] << " ";
	}
}
```

코드 출력 결과



Complexity 분석

비교와 삽입 횟수를 측정했기에, 어렵지 않게 알 수 있다.

<Insertion Sort>

for 문 안의 while 문 속 insertionCompare++; insertionSwap++;을 보면 알 수 있듯, 최악의 경우 O(n^2)의 시간복잡도.

Time Complexity : O(n^2)

Space Complexity : O(n)

<Quick Sort>

Partition Func 를 보면 알 수 있듯, 1 번의 partition 실행 시, parameter 로 들어온 l, r 에 대해

약 r-l 회 정도의 비교 연산을 수행하게 됨.

총 input 배열의 길이를 n 이라 하면, 이는 순환 호출 1 회 당 약 O(n)의 시간복잡도를 갖는다 할 수 있음.

따라서 값이 한쪽으로 몰려있지 않아 pivot 값을 제대로 잡을 수 있는 대부분의 경우 (최선 or 평균적으로) 순환 호출의 깊이가 log(n)이라 할 수 있으므로,

최선/평균의 시간복잡도는 O(nlogn).

값이 한쪽으로 몰려있는 등, pivot 값을 잘 잡지 못해 partition 을 한쪽으로 쏠려 진행하게 되는 경우(최악)의 순환 호출 깊이는 약 n-k 회(n 회) 라 할 수 있으므로,

최악의 시간복잡도는 O(n^2)

Time Complexity : Best - O(nlogn) / Average – O(nlogn) / Worst – O(n^2)

Space Complexity : O(n)

<Heap Sort>

Heap 은 Complete Binary Tree 이므로, 자료가 한 쪽에 몰려 Tree 의 깊이가 비정상적으로 깊어질 일이 없다. 따라서 자료 n 개에 대하여 깊이는 약 log(n)이라 할 수 있음.

MakeHeap(Heapify)를 모든 자료에 실행하면 Heap 이 완성되는데, 그렇게 하면 정렬이 끝나므로,

Heapify 의 시간복잡도만 측정하면 해결된다.

Heapify 의 시간복잡도는, Tree 의 깊이에 따라 결정되는데, 깊이가 log(n)이므로, 시간복잡도도 O(logn).

따라서 전체 시간복잡도는 n * O(logn) -> O(nlogn)

Time Complexity : O(nlogn)

Space Complexity : O(n)