HW8 REPORT

학번: B911026

이름 : 김민욱

(1) Select sort 구현 코드

```
void selectionsort(int arr[],int n) {
    int i, j, min, temp, indexOfMin;

    for (i = 0; i < n - 1; i++)
    {
        min = arr[i];
        indexOfMin = i;
        for (j = i + 1; j < n; j++)
        {
            if (min > arr[j]){
                min = arr[j];
                indexOfMin = j;
            }
            C1++;
        }
        temp = arr[i];
        C2++;
        arr[i] = arr[indexOfMin];
        C2++;
        arr[indexOfMin] = temp;
    }

}
```

(2) Insertion sort 구현 코드

```
void insertionsort(int arr[],int n){
    int i, j, key;
    for (i = 1; i < n; i++){
        key = arr[i];

        for (j = i - 1; j >= 0 && arr[j] > key; j--){
            arr[j + 1] = arr[j];
            C1++;
            C2++;
        }
        arr[j + 1] = key;
        C2++;
    }
}
```

(3) Quick sort 구현 코드

```
void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}

int partition(int arr[], int l, int r){
    int x = arr[l];
    int i = l + 1;
    int j = r;

    while(1){

      while (arr[i] <= x)
      {
        i++;
        C1++;
      }
      C1++;
      while (arr[j] > x)
      {
        j--;
        C1++;
      }
      C1++;
      if (i < j)
      {
        swap(&arr[i], &arr[j]);
        C2+=2;
        i++;
        j--;
      }
      else
      {
        break;
      }
    }
    swap(&arr[l], &arr[j]);
    C2+=2;
    return j;
}
```

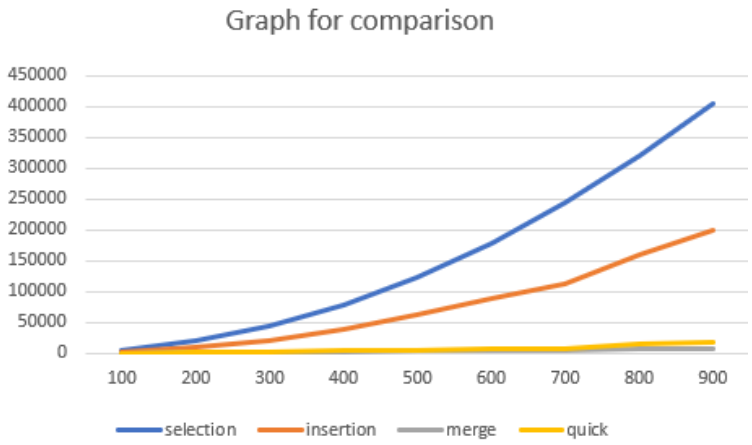
```
void quicksort(int arr[], int l, int r){
    int p;
    if (l < r){
        p = partition(arr, l, r);
        quicksort(arr, l, p - 1);
        quicksort(arr, p + 1, r);
    }
}
```

```
void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}
```

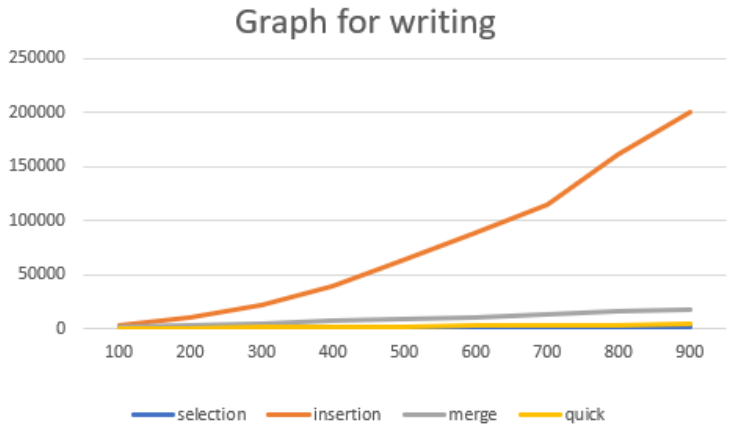
결과

```
** Selection Sort **
n = 100; C1 = 4950; C2 = 198.
n = 200; C1 = 19900; C2 = 398.
n = 300; C1 = 44850; C2 = 598.
n = 400; C1 = 79800; C2 = 798.
n = 500; C1 = 124750; C2 = 998.
n = 600; C1 = 179700; C2 = 1198.
n = 700; C1 = 244650; C2 = 1398.
n = 800; C1 = 319600; C2 = 1598.
n = 900; C1 = 404550; C2 = 1798.

** Insertion Sort **
n = 100; C1 = 2643; C2 = 2742.
n = 200; C1 = 10507; C2 = 10706.
n = 300; C1 = 21685; C2 = 21984.
n = 400; C1 = 39027; C2 = 39426.
n = 500; C1 = 63237; C2 = 63736.
n = 600; C1 = 88097; C2 = 88696.
n = 700; C1 = 113886; C2 = 114585.
n = 800; C1 = 160128; C2 = 160927.
n = 900; C1 = 198991; C2 = 199890.

** Merge Sort **
n = 100; C1 = 535; C2 = 1344.
n = 200; C1 = 1290; C2 = 3088.
n = 300; C1 = 2096; C2 = 4976.
n = 400; C1 = 2965; C2 = 6976.
n = 500; C1 = 3865; C2 = 8976.
n = 600; C1 = 4796; C2 = 11152.
n = 700; C1 = 5767; C2 = 13352.
n = 800; C1 = 6691; C2 = 15552.
n = 900; C1 = 7741; C2 = 17752.

** Quick Sort **
n = 100; C1 = 711; C2 = 352.
n = 200; C1 = 1881; C2 = 730.
n = 300; C1 = 3635; C2 = 1202.
n = 400; C1 = 4590; C2 = 1638.
n = 500; C1 = 6169; C2 = 2118.
n = 600; C1 = 7086; C2 = 2688.
n = 700; C1 = 7795; C2 = 3206.
n = 800; C1 = 16950; C2 = 3634.
n = 900; C1 = 17882; C2 = 4236.
```

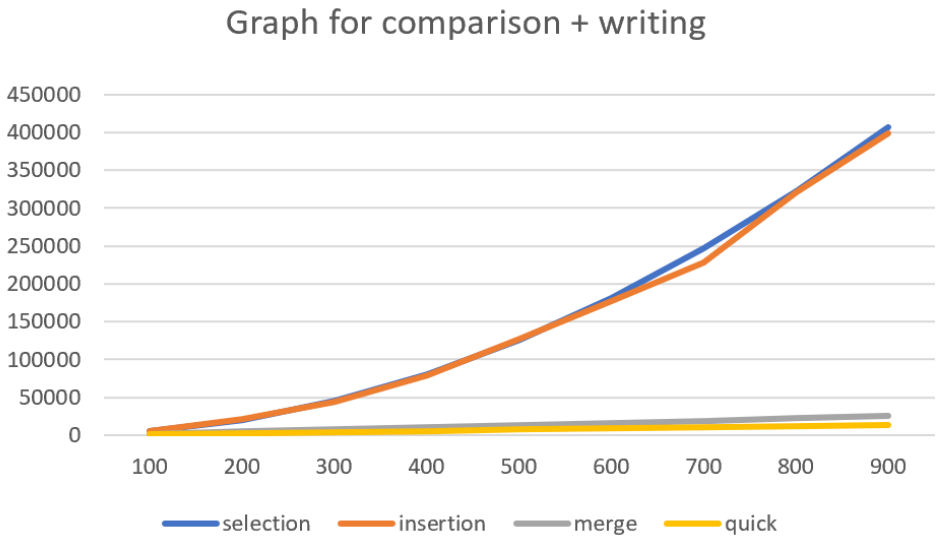
1. 키값 비교 연산 횟수에 대한 그래프



2. 쓰기 연산 횟수에 대한 그래프



3. 총 연산 횟수에 대한 그래프

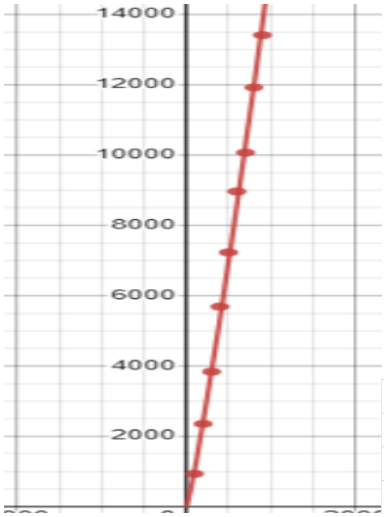


분석

(a) 삽입 정렬에서 키 값 비교는 정확히 $\frac{N*(N-1)}{2}$번 일어나기 때문에 키 값 비교 횟수가 $\frac{N^2}{4}$에 비례하는 선택 정렬에 비해 그래프가 더욱 빠르게 증가하는 것을 볼 수 있다. 하지만 삽입 정렬은 쓰기 연산 횟수가 $2N-2$에 비례하고, 선택 정렬은 $\frac{N^2}{4}$에 비례하기 때문에 최종 성능(키 값 비교 횟수 + 쓰기 연산 횟수)를 비교한다면 두 정렬 알고리즘의 성능이 거의 비슷하다는 것을 알 수 있다. 또한 대부분의 경우에 쓰기 연산이 비교 연산보다 expensive (more overhead) 하기 때문에 어떤 경우엔 오히려 선택 정렬이 더 빠르다고 생각해 볼 수도 있다.

(b) Big-O 표기법에서 충분히 큰 n에 대해서 $f(n) \leq C * g(n)$을 만족하는 C가 존재하면 우리는 $f(n) = O(g(n))$ 라고 표현한다. 따라서 퀵 소트의 최종 성능 그래프가 $C * n * log(n)$보다 느리게 증가하는 C값을 구한다면 우리는 퀵 소트가 O(nlogn) 시간 복잡도에서 작동한다고 말할 수 있다.

다음은 각 N에 대한 C값과 C=1.55로 설정한 그래프이다.



	100	200	300	400	500	600	700	800	900
	931	2351	3838	5688	7231	8964	10071	11926	13414
n*logn	664.3856	1528.771	2468.646	3457.542	4482.892	5537.291	6615.848	7715.085	8832.403
C	1.401295	1.537836	1.554699	1.645099	1.613021	1.618842	1.522254	1.545803	1.518726

퀵 소트에게 있어 최악의 성능을 내는 경우는 이미 정렬된 경우이다. 왜냐하면 partition 함수가 작동하면 pivot을 기준으로 배열을 partition 하게 되는데 배열이 이미 정렬되어 있다면 pivot을 제외한 모든 원소들이 pivot보다 크거나 작기 때문에 순환 호출의 깊이가 n이 되게 된다. 또한 각 단계에서 비교 연산이 n번 일어나므로 퀵 소트의 최악의 시간복잡도는 $O(N^2)$이다. 하지만 N개 원소를 가진 배열이 있을 때 이미 정렬되어 있을 확률은 $\frac{2}{N!}$이기 때문에 사실상 불가능에 가깝다.

(c)퀵 정렬은 키 값 비교 연산 횟수가 합병 정렬에 비해 약 1.2배 정도 더 많다. 하지만 대입 연산 횟수는 합병 정렬이 퀵 정렬보다 대략 4배 정도 많은 것을 확인할 수 있다. 따라서 최종 성능(키 값 비교 횟수 + 쓰기 연산 횟수)를 비교한다면 병합 정렬이 약 2배 정도 연산 횟수가 많다. 또한 퀵 정렬의 경우 별도의 공간을 할당하지 않아도 되는 in-place 구현이 가능하지만 병합 정렬의 경우 merge() 함수를 호출할 때마다 별도의 배열을 할당해 이 배열에 원소들을 복사하고 다시 원래 있던 배열로 복사하는 과정이 필요하다. 또한 퀵 정렬은 비교 연산을 pivot을 기준으로 실행하기 때문에 cache를 사용해 조금 더 빠르게 연산이 가능하지만 합병 정렬은 매번 비교하는 피연산자가 바뀌므로 퀵 정렬 보다 느리다..따라서 병합 정렬은 퀵정렬 보다 (expensive한) 대입 연산이 많고 이 배열을 생성할 때 드는 시간 때문에 퀵 정렬보다 느릴 수 밖에 없다.