Zijian Gao
26319560
2/29/2016
CS311 HW2

8-1-4.)

A sequence of n elements with n / k subsequences, with k elements per sequence, each individual subsequences, if sorted with an efficient comparison-based sort such as mergesort, requires log(k) comparisons to sort, as this is the lower bound on the amount of comparisons required to sort an equivalently sized list. Since it is given that there are n such subsequences, and that they are already sorted relative to each other, the fewest amount of iterations that must be made by a sort is n, the amount of subsequences, as every subsequence will be sorted element-by-element using the fewest possible comparisons. Since this is n times log(k) comparisons, the sequence takes $\Omega(n \log(k))$ comparisons to sort.

8-2-4.)

```
procedure generate_database(A[], n, k):
    int[] database
    database = [k + 1] //k + 1 total elements, each element is init to 0
    for (i from 0 to n)
        database[A[i]] += 1

    for (i from 1 to k)
        database[i] += database[i - 1]

    return database

procedure query_range(database, a,b):
    if (database == NULL || b > k || a < 0)
        error

    if (a == 0)
        return database[b]
    else
        return database[b] - [database[a - 1]]
```

This algorithm is called first on generate_database, which creates an empty array of k + 1 elements, each initialized to the value 0. Then, it iterates through the input array A and increments the count in the database for the i'th element. This consumes $\Theta(n)$ time. Next, from 1 to k, the counts for the elements are summed from the range 0 to i, such that at each j index position 0 to k for database[], the value for that index represents the occurences of integers from 0 to j. This trivially consumes $\theta(k)$ time for a total of $\theta(n+k)$ time.

After generate_database is called, query_range can be used, passing in the generated array and a range (a,b) inclusive. If the beginning of the range is 0, then only the ending is required as the database is expected to have the total counts for integers 0 through b at the b'th index position. Otherwise, take the count for integers from 0 through b, and subtract the count for integers from 0 to a-1, so that the remaining range is from a to b. Both of the return paths for this method run in constant time.