1 Hashing Asymptotics

Suppose we set the `hashCode` and `equals` methods of the `ArrayList` class as follows.

```java
/* Returns true iff the lists have the same elements in the same ordering */
@override
public boolean equals(Object o) {
    if (o == null || o.getClass() != this.getClass() || o.size() != this.size()) {
        return false;
    }
    ArrayList<T> other = (ArrayList<T>) o;
    for (int i = 0; i < this.size(); i++) {
        if (other.get(i) != this.get(i)) {
            return false;
        }
    }
    return true;
}

/* Returns the sum of the hashCodes in the list. Assume the sum is a cached instance variable. */
@override
public int hashCode() {
    return sum;
}
```

(a) Give the best and worst case runtime of `hashContents` in $\Theta(.)$ notation as a function of $N$, where $N$ is initial size of the list. Assume the length of `set`'s underlying array is $N$ and the `set` does not resize. Assume the `hashCode` of an `Integer` is itself. Admittedly, the `ArrayList` class does not have the method `removeLast`, but assume it does for this problem, and is implemented the same as in Project 2b. Finally, assume $f$ accepts two `int`s, returns an unknown `int`, and runs in constant time.

```java
static void hashContents(HashSet<ArrayList<Integer>> set, ArrayList<Integer> list) {
    if (list.size() <= 1) {
        return;
    }
    int last = list.removeLast();
    list.set(0, f(list.get(0), last));
    set.add(list);
    hashContents(set, list);
}
```

Best Case: $\Theta(N)$, Worst Case: $\Theta(N^2)$
**Add:**
1. Compute hashcode $\rightarrow$ object.hashcode()
2. Go to correct bucket $\rightarrow$ hashcode % M = bucket
3. [Bucket through linked list]
   - If equal key, return
   - Else: insert at end of bucket

Two possibilities:

**Worst Case:**
1. go to bucket 0: $O(N)$
2. go to another

**Best Case:**
- bucket (that is empty): $O(1)$

**Worst Case:**
\[
\sum_{i=0}^{N-2} i \rightarrow N + 2 + 3 + 4 + \ldots + N - 1
\]
\[
\sum_{i=1}^{N-2} i \rightarrow \frac{N(N-1)}{2}
\]
\[
O(N^2)
\]

**Best Case:**
\[
1 + 1 + \ldots + 1
\]
\[
O(N)
\]
(b) Continuing from the previous part, how can we define \( f \) to ensure the worst case runtime? How can we define \( f \) to ensure the best case runtime? There may be multiple possible answers.

1. Worst case:

```java
1    int f(int first, int last) {
2         return __________________________;
3    }
```

2. Best case:

```java
1    int f(int first, int last) {
2         return __________________________;
3    }
```
2 Sorted Runtimes

We want to sort an array of $N$ unique numbers in ascending order. Determine the best case and worst case runtimes of the following sorts:

(a) Once the runs in merge sort are of size $< N/100$, we perform insertion sort on them.
   
   Best Case: $\Theta(\sqrt{N})$, Worst Case: $\Theta(N^3)$

(b) We can only swap adjacent elements in selection sort.
   
   Best Case: $\Theta(N^2)$, Worst Case: $\Theta(N^2)$

(c) We use a linear time median finding algorithm to select the pivot in quicksort.
   
   Best Case: $\Theta(N\log N)$, Worst Case: $\Theta(N\log N)$

(d) We implement heapsort with a min-heap instead of a max-heap. You may modify heapsort but must maintain constant space complexity.
   
   Best Case: $\Theta(N\log N)$, Worst Case: $\Theta(N\log N)$

(e) We run an optimal sorting algorithm of our choosing knowing:
   
   • There are at most $N$ inversions
     Best Case: $\Theta(N)$, Worst Case: $\Theta(N)$
   
   • There is exactly 1 inversion
     Best Case: $\Theta(1)$, Worst Case: $\Theta(N)$
   
   • There are exactly $(N^2 - N)/2$ inversions
     Best Case: $\Theta(\ldots)$, Worst Case: $\Theta(\ldots)$

Recall an inversion is an "out of place" pair. For instance, in the array:

\[
\begin{array}{ccccccccc}
3 & 2 & 7 & 6 & 4 & 1 & 5 & 0 \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7
\end{array}
\]

- $(2,0)$ is an inversion since 2 precedes 0
- $(2,4)$ is not an inversion since 2 and 4 are in order

Running an insertion sort is $\Theta(N^2)$

- $\Theta(N + N)$
  \begin{align*}
  \sim & \Theta(N) \\
  \sim & \Theta(\text{Inversions})
  \end{align*}