Assignment 2

Quinn Perfetto, 104026025
60-454 Design and Analysis of Algorithms

February 14, 2017

Question 1 (a).

Algorithm 1: FlipSort(L, lower, upper)

Input: $L[lower..upper], lower \leq i \leq upper,\ L[i] \in \{0, 1\}$
Output: $L[lower..upper]$ sorted in ascending order

begin
    if *upper - lower > 1* **then**
        FlipSort(L, lower, $\lfloor \frac{lower+upper}{2} \rfloor$);
        FlipSort(L, $\lfloor \frac{lower+upper}{2} \rfloor + 1$, upper);
        return *Merge*($L\left[lower..\lfloor \frac{lower+upper}{2} \rfloor\right]$, $L\left[\lfloor \frac{lower+upper}{2} \rfloor + 1..upper\right]$)

Algorithm 2: Merge(A, B)

Input: Two sorted lists $A[1..n]$ and $B[1..m]$ over $\{0, 1\}$
Output: A sorted list $C[1..m+n]$ containing all elements of A and B
Let $<>$ be the list concatenation operator

begin
    $index_A$:= SequentialSearch(A, 1);
    $index_B$:= SequentialSearch(B, 1);
    if $index_A == 0$ **then**
        return $A <> B$
    if $index_B == 0$ **then**
        return $B <> A$
    return
    $A[1..index_A - 1] <> flip(A[index_A..n] <> B[1..index_B - 1]) <> B[index_B..m]$

Lemma 1.1. Algorithm Merge correctly produces a sorted list

Note that SequentialSearch(L, x) refers to the algorithm defined in Chapter 0 Page 10 of the CourseWare, and returns the index of the first occurrence of x in L if it exists, and 0 otherwise. For the sake of brevity, we take $L[1..0]$ as [] (the empty list).

Case 1: $index_A == 0$
$index_A == 0 \Rightarrow A$ contains no instance of 1, i.e. $1 \leq i \leq n,\ A[i] == 0$. Since $0 \leq 0 \leq 1$ and B is assumed to be a sorted list over $\{0,1\}$, by the transitivity of $\leq$ $A <> B$ must also be sorted. Thus the algorithm works correctly.

Case 2: $index_B == 0$
This case is similar to the above case, I thus omit the detail.

Case 3: $index_A \geq 1 \wedge index_B \geq 1$
Without loss of generality, let $A = 0^x 1^{n-x}$ and $B = 0^y 1^{m-y}$ such that $x, y \geq 0,\ x \leq n,\ y \leq m$. Since $index_A$ refers to the first occurrence of 1 in A, $A[1..index_A - 1] = 0^x$ and $A[index_A..n] = 1^{n-x}$. By a similar argument for $index_B$, $B[1..index_B - 1] = 0^y$ and $B[index_B..m] = 1^{m-y}$. Thus,

$$\begin{aligned}
&A[1..index_A - 1] <> flip(A[index_A..n] <> B[1..index_B - 1]) <> B[index_B..m] \\
&= 0^x <> flip(1^{n-x}, 0^y) <> 1^{m-y} \\
&= 0^x <> (0^y <> 1^{n-x}) <> 1^{m-y} \\
&= 0^x 0^y 1^{n-x} 1^{m-y}
\end{aligned}$$

Which is a sorted list of length $x + y + n - x + m - y = n + m$. Therefore the algorithm works correctly.

Lemma 1.2. Algorithm FlipSort correctly produces a sorted list.

Induction on the size of the input n.
(Induction Basis) If $n = 1$ FlipSort performs no operations and returns a single element list which is vacuously sorted.
(Induction Hypothesis) Assume that FlipSort correctly sorts all lists of size $n \leq k,\ n > 1$.
(Induction Step) Let $L'[lower..upper]$ be a list of length $k+1$, i.e. $upper - lower = k + 1$. The first recursive call produces a list of length,

$$\begin{aligned}
&\lfloor \frac{lower + upper}{2} \rfloor - lower \\
&\leq \frac{lower + upper}{2} - lower \\
&= \frac{upper - lower}{2} \\
&< upper - lower && (n > 1) \\
&= k + 1
\end{aligned}$$

Thus by the Inductive Assumption the first recursive call produces a correctly sorted list $L'[lower..\lfloor \frac{lower+upper}{2} \rfloor]$ (I).

Additionally the second recursive call produces a list of length,

$$
\begin{aligned}
& upper - \lfloor \frac{lower + upper}{2} \rfloor + 1 \\
& \leq upper - \frac{lower + upper}{2} + 1 \\
& = \frac{upper - lower}{2} + 1 \\
& = \frac{k+1}{2} + 1 \\
& < k + 1 && (k + 1 > 2)
\end{aligned}
$$

Thus by the Inductive Assumption the second recursive call produces a correctly sorted list $L'[\lfloor \frac{lower+upper}{2} \rfloor..upper]$ (II).

Finally by Lemma 1.1, (I) and (II) we know that the Merge Algorithm correctly merges the resulting lists into a sorted list $L'[lower..upper]$.
Therefore Algorithm FlipSort works correctly.

Lemma 1.3. Algorithm Merge requires at most $2n + 2m$ operations

As proved in the CourseWare SequentialSearch search performs at most n comparisons for $index_A$ and at most m comparisons for $index_B$. Further, since $Flip$ requires $O(j - i)$ time and $j - i \leq n + m$ we have at most $(n + m) + (n + m) = 2n + 2m$ operations.

Lemma 1.4. Algorithm FlipSort is $\theta(nlgn)$

Let $T(n)$ be the time required to sort a list of n elements with FlipSort.

$$
T(n) = \begin{cases} T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 2n & n > 1 \\ 0 & otherwise \end{cases}
$$

Let $T_{\lfloor\rfloor}(n) = 2T(\lfloor \frac{n}{2} \rfloor) + 2n$ and $T_{\lceil\rceil}(n) = 2T(\lceil \frac{n}{2} \rceil) + 2n$.
Using the general formula for solving recurrences, we have
$f(n) = 2n = \theta(n) = \theta(n^{log_2 2}) = \theta(n^{log_b a} lg^0 n)$

Therefore $T_{\lfloor\rfloor}(n) = T_{\lceil\rceil}(n) = \theta(nlgn)$.
Then $T_{\lfloor\rfloor}(n) \leq T(n) \leq T_{\lceil\rceil}(n) \Rightarrow T(n) = \theta(nlgn)$.

Therefore Algorithm FlipSort correctly sorts the input and runs in $\theta(nlgn)$ time. ■

Question 1 (b).