ITM 613 Algorithms

Mid Term Assignment Solutions

Gradings:

- Give full points when correct, 1/n for solving each n subproblems. 0 for totally wrong or none, -1 for each errors.
- There may be partial points for proofs if the direction is correct.

1. [Soundex] (5 pts)

Soundex is a phonetic algorithm for indexing names by sound, as pronounced in English. It is used for encoding similar words into a same representation. It was first developed in 1910s, even before computers were in use.

The Soundex code for a name consists of a letter followed by three numerical digits. The algorithm is as follows:

1. Save the first letter of the name and drop all other occurrences of `[a, e, i, o, u, y, h, w]`.
2. Replace consonants with digits as follows (after the first letter):
   - `b, f, p, v` → 1
   - `c, g, j, k, q, s, x, z` → 2
   - `d, t` → 3
   - `l` → 4
   - `m, n` → 5
   - `r` → 6
3. If two or more letters with the same number are adjacent in the original name (before step 1), only leave the first letter; also two letters with the same number separated by 'h' or 'w' are coded as a single number, whereas such letters separated by a vowel are coded twice. This rule also applies to the first letter.
4. If you have too few letters in your word that you can't assign three numbers, append with zeros(`0`) until there are three numbers. If you have four or more numbers, retain only the first three.

For example, applying "`algorithm`" with the algorithm, the result for each steps are as follows:

1. `algrtm`
2. `a42635`
3. `a42635`

4. `a426`

Hence we have `a426` for the word "`algorithm`".

Similarly, we can encode "`janghwan`" to `j525`.

- (a) Apply the name "`alkhwarizmi`", a 8c Persian mathematician, with the algorithm. Show results for each steps as above.
- (b) Apply your own name with the algorithm, showing results for each steps above.

[Solutions]

- (a) `A426`
- (b) differs by each student. For "janghwan", `j525`

2. [Asymptotic notations] (9 pts)

State whether the following statements are true or false, and prove or disprove it. Explain your answer. (3 pts each)

(a) $\frac{1}{2}n^2 - 3n = \Theta(n^2)$

(b) $6n^3 = \Theta(n^2)$

(c) $an^2 + bn + c = \Theta(n^2)$

[Solutions]

(a) True. We must determine positive constants c_1, c_2, and n_0 such that

$$c_1 n^2 \le \frac{1}{2}n^2 - 3n \le c_2 n^2$$

for all $n \ge n_0$. Dividing by n^2 yields

$$c_1 \le \frac{1}{2} - \frac{3}{n} \le c_2$$

We can make the right-hand inequality hold for any value of $n \ge 1$ by choosing any constant $c_2 \ge 1/2$. Likewise, we can make the left-hand inequality hold for any value of $n \ge 7$ by choosing any constant $c_1 \ge 1/14$. Thus, by choosing $c_1 = 1/14$, $c_2 = 1/2$, and $n_0 = 7$, we can verify that $\frac{1}{2}n^2 - 3n = \Theta(n^2)$.

(b) False. Suppose for the purpose of contradiction that c_2 and n_0 exist such that $6n^3 \leq c_2 n^2$ for all $n \geq n_0$. But then dividing by n^2 yields $n \leq c_2/6$, which cannot possibly hold for arbitrarily large n, since c_2 is constant.

(c) True. Throwing away the lower-order terms and ignoring the constant yields $f(n) = \Theta(n^2)$.

Formally, we take the constants $c_1 = a/4, c_2 = 7a/4$, and $n_0 = 2\max(|b|/a, \sqrt{|c|/a})$. We can verify that $0 \leq c_1 n^2 \leq an^2 + bn + c \leq c_2$ for all $n \geq n_0$.

3. [Search] (40 pts)

Consider the *searching problem*:

- **Input**: A sequence of n numbers $A = \langle a_1 . a_2, \ldots, a_n \rangle$ and a value v
- **Output**: An index i such that $v = A[i]$ or the special value NIL if v does not appear in A.

(a) Write pseudocode for `LINEAR-SEARCH(A, v)`, which scans through the sequence, looking for v. *(10 pts)*

(b) Using a loop invariant, prove that your algorithm is correct. Make sure that your loop invariant fulfills the three necessary properties. *(5 pts)*

(c) Write pseudocode for linear search using the divide-and-conquer approach. *(10 pts)*

[binary search]

Here is a new observation: If sequence is already sorted, we can check the midpoint value of sequence with , and ignore half of the sequence. This *binary search algorithm* may be repeated until is found, or the sequence becomes emtpy. It halves the size of the remaining sequence each time of division.

(d) Write pseudocode, either iterative or recursive, for binary search. Argue that the worst-case running time of binary search is $\Theta(\lg n)$. *(10 pts)*

(e) Use the master method to show that the soultion to the binary-search recurrence $T(n) = T(n/2) + \Theta(1)$ is $T(n) = \Theta(\lg n)$. *(5 pts)*

[Solutions]

(a)

```
LINEAR-SEARCH($A, v$)
1  for $i = 1$ to $A.length$
2      if $A[i] == v$
3          return $i$
4  return NIL
```

(b)

Loop invariant: At the start of each iteration of the **for** loop, the subarray $A[1..i-1]$ consists of elements that are different than v.

- **Initialization:** Initially the subarray is the empty array, so the prove is trivial.
- **Maintenance:** On each step, we know that $A[1..i-1]$ does not contain v. We compare it with $A[i]$. If they are the same, we return i, which is a correct result. Otherwise, we continue to the next step. We have already insured that $A[1..i-1]$ does not contain v and that $A[i]$ is different from v, so this step preserves the invariant.
- **Termination:** The loop terminates when $i > A.length$. Since i increases by 1 and $i > A.length$, we know that all the elements in A have been checked and it has been found that v is not among them. Thus, we return `NIL`.

(c)

```
RECURSIVE-LINEAR-SEARCH$(A, v, i, j)$
if $i \geq j$
    if $A[i] == v$
        return $i$
    else
        return NIL
// split the problem into half
$i1 =$ RECURSIVE-LINEAR-SEARCH$(A, v, i, \lfloor j/2 \rfloor)$
$i2 =$ RECURSIVE-LINEAR-SEARCH$(A, v, \lfloor j/2 \rfloor + 1, j)$
// combine results
if $i1 \neq$ NIL
    return $i1$
else
    return $i2$
```

(d)

```
RECURSIVE-BINARY-SEARCH$(A, v, low, high)$
if $low > high$
    return NIL
$mid = \lfloor (low + high)/2 \rfloor$
if $v == A[mid]$
    return $mid$
elseif $v > A[mid]$
    return RECURSIVE-BINARY-SEARCH$(A, v, mid + 1, high)$
else
    return RECURSIVE-BINARY-SEARCH$(A, v, low, mid - 1)$
```

Each time we do the comparison of v with the middle element, the search range continues with range halved.

The recurrence

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ T(n/2) + \Theta(1) & \text{if } n > 1. \end{cases}$$

The solution of the recurrence is $T(n) = \Theta(\lg n)$.

(e)

$$\begin{aligned} a &= 1, b = 2, \\ f(n) &= \Theta(n^{\lg 1}) = \Theta(1), \\ T(n) &= \Theta(\lg n). \end{aligned}$$

4. [Maximum Subarray problem] (10 pts)

Use the following ideas to develop a nonrecursive, linear-time algorithm for the maximum-subarray problem: Start at the left end of the array, and progress toward the right, keeping track of the maximum subarray seen so far.

Knowing a maximum subarray $A[1..j]$, extend the answer to find a maximum subarray ending at index $j+1$ by using the following observation:

- a maximum subarray $A[i\,..\,j+1]$, for some $1 \le i \le j+1$

Determine a maximum subarray of the form $A[i\,..\,j+1]$ in constant time based on knowing a maximum subarray ending at index j.

(hint: Use dynamic programming)

[Solution]

```
ITERATIVE-FIND-MAXIMUM-SUBARRAY($A$)
$n = A.length$
$max\text{-}sum = -\infty$
$sum = -\infty$
for $j = 1$ to $n$
    $current\text{-}high = j$
    if $sum > 0$
        $sum = sum + A[j]$
    else
        $current\text{-}low = j$
        $sum = A[j]$
    if $sum > max\text{-}sum$
        $max\text{-}sum = sum$
        $low = current\text{-}low$
        $high = current\text{-}high$
return $(low, high, max\text{-}sum)$
```