CM2035: Algorithms & Data Structures II

Midterm Assignment

Arjun Muralidharan

3rd July 2021

Contents

List of Figures

List of Tables

List of Listings

1 Task 1

1.1 Question 1

When the array A is distributed to two other arrays, the even indices of the values in A are halved and floored, i.e. the index $A[6]$ becomes $A1[3]$ and a value at $A[3]$ becomes $A2[1]$ (because 3 divided by two is 1.5, which is 1 when rounded down).

This means that when we find a matching key, we need to return the doubled value $2 * i$ to get the original index. **The value j represents whether the input array B is storing the even or odd indices of A.** If j is 0, we consider B to contain even indices of A. If j is 1, we consider B to store odd indices of A.

1.2 Question 2

The recursive version of the FIND function is shown in Algorithm 1.

Algorithm 1 Recursive FIND function

```
function RECURSIVEFIND($key, B, M, i, j$)
    if $i < 0$ then
        return $-1$
    end if
    if $B[i] == key$ then
        return $j + 2 * i$
    end if
    return FIND($key, B, M, i - 1, j$)
end function
```

1.3 Question 3

The master theorem applies to recurrence equations of the form

$$T(n) = a * T(\frac{n}{b}) + f(n)$$

Since the recurrence equation $T(M) = T(M - 2) + C$ does not express $T(M)$ in terms of a fraction of M, i.e., as $T(\frac{M}{b})$ where $b > 1$, the master theorem does not apply.

2 Task 2

2.1 Question 1

The **best case** is given when the desired key is present in the *first* element of the *first* subarray $A1$. In the provided example, the inputs would be

$$key = 7, A1 = [7, 6, 2, 4, 9], A2 = [5, 3, 1, 8], N = 9$$

while the worst case inputs are given if the desired key is the *last* element in the *second* subarray $A2$, with the inputs

$$key = 8, A1 = [7, 6, 2, 4, 9], A2 = [5, 3, 1, 8], N = 9$$

2.2 Question 2

First, we'll analyse the running time of the Find function by using the **method of counting up time units**.

Algorithm 2 Worst case time complexity of the Find function

```
function Find($key, B, M, j$)
    for $0 \leq i < M$ do          // 7N+5 time units
        if $B[i] == key$ then      // 5 time units
            return $j + 2 * i$     // not executed in the worst case
        end if
    end for
    return $-1$                    // 1 time unit
end function
```

Algorithm 3 Best case time complexity of the Find function

```
function Find($key, B, M, j$)
    for $0 \leq i < M$ do          // 4 time units
        if $B[i] == key$ then      // 5 time units
            return $j + 2 * i$     // 5 time unites
        end if
    end for
    return $-1$                    // not executed in the best case
end function
```

The worst case running time as shown in Algorithm 2 can be expressed as $T(N) = 7N + 11$. The best case can be expressed as $T(N) = 14$, as shown in Algorithm 3.

Algorithm 4 Worst case time complexity of the R0 function

```
function R0($key, A1, A2, N$)
    index = FIND($key, A1, \lceil \frac{N}{2} \rceil, 0$)    // $7N + 11 + 1$
    if index == -1 then    // 2 time units
        return FIND($key, A1, \lfloor \frac{N}{2} \rfloor, 1$)    // $7N + 11 + 1$
    end if
    return index    // not executed in the worst case
end function
```

Algorithm 5 Best case time complexity of the R0 function

```
function R0($key, A1, A2, N$)
    index = FIND($key, A1, \lceil \frac{N}{2} \rceil, 0$)    // 14 time units
    if index == -1 then    // not executed
        return FIND($key, A1, \lfloor \frac{N}{2} \rfloor, 1$)    // not executed
    end if
    return index    // 2 time units
end function
```

The worst case time complexity of R0 can be stated as $T(N) = 7N + 11 + 1 + 2 + 7N + 11 + 1 = 14N + 27$, as shown in Algorithm 4. The best case time complexity is given as $T(N) = 16$, as shown in Algorithm 5.

2.3 Question 3

We can use asymptotic analysis to simplify the running times and express them as growth functions.

In the worst case, the FIND function needs to loop over both subarrays, it's running time is linear to the input size, $\mathcal{O}(N) = N$. In the best case, the function needs to only inspect the first element of the first subarray, hence the running time is constant $\Omega(N) = C$.

2.4 Question 4

Skipped.

3 Task 3

3.1 Question 1

Algorithm 6 Pseudocode for R1 algorithm

```
function R0($key, A, B, N$)
    $j = -1$
    for $0 \leq i$ ¡ N do
        if $A[i] == key \wedge B[i] == key$ then
            $j = i$
        end if
        if $A[i] \neq B[i]$ then
            return $-2$
        end if
    end for
    return $j$
end function
```

3.2 Question 2

The time complexity can be noted in Theta notation as $\Theta(N)$, as the function always needs to traverse the entire array to check for consistency. So even if the key is found, the algorithm needs to check every pair of elements in both arrays every time the algorithm runs. Hence the time complexity is proportional to the size of the input.

4 Task 4

4.1 Question 1

Algorithm 7 Pseudocode for R1Hash Search algorithm

```
function R1Hash($key, a, b, A, B, N$)
    hashPos = $(a * key + b)$ mod $N^2$
    index = $B[hashPos]$
    if A[index] == $key$ $\vee$ index $< 0$ then
        return index
    end if
    return -2
end function
```

4.2 Question 2

The three cases are handled by the pseudocode in Algorithm 7 as follows.

Case 1: Value found In the algorithm, the position of the hashed value in B is calculated and stored in *hashPos*. The index of the value in A is then read from B at the index *hashPos*. If the value in A at this index is equal to the *key*, we return the index with the correct position as desired.

Case 2: Value not found If the index is less than 0, we can assume that it is -1, and hence the value was not found, and we can return the index as -1.

Case 3: Error in storage If the index is non-negative, but didn't match the key, then we return -2 as the fallback to indicate an error.

5 Task 5

The setting we will use is **direct referencing** where an array A's indices are stored in an array B at the index $B[A[i]]$. The length of B is the maximum value stored in A, and B is filled with -1 by default.

5.1 Question 1

Algorithm 8 Pseudocode for Direct Reference Search

```
function DirectRefSearch($key, A, B$)
    if $B[key] == A[B[key]] \quad \vee \quad B[key] < 0$ then
        return $B[key]$
    end if
    return -2
end function
```

5.2 Question 2

The algorithm in Algorithm 8 checks if the value at the index in B that corresponds to *key* is equal to the corresponding value in A, and returns the index. It also returns the index if the value in B is less than 0, in which case it can be assumed to return -1.

If these checks fail, the algorithm returns -2.

5.3 Question 3

The best case and worst case are practically identical as no iteration is needed in this setup. The cost of this is that the solution can take a lot of space. Hence the best case here would be an Array A where the largest number is 1 and the length of the array is 1. The worst case would be an array where the largest number is very, very large. However none of these inputs impact time complexity of the search algorithm.

5.4 Question 4

The best and worst case running times are always $T(N) = C$.

5.5 Question 5

The Theta notation for this algorithm is $\Theta(1)$.