Task 1: Consider the following pseudocode function:

function Find(key,B,M,j)

for 0 <= i < M

if(B[i] == key):

return j + 2\*i

return -1

1. This function takes an array B, its length M, a non-negative integer key, and a non-negative integer j as inputs, and returns an integer. Briefly explain why if the input array B is one of the arrays (A1 or A2) created as above, this function will return the index where the value key is located in the original array A. In your explanation refer to the input j, and to what it corresponds in array A. [5 marks]

This algorithm cycles through array B and searches for the value that is equal to the key value. It returns the index of this value in A (j + 2\*i), based on the position inside B (i), or returns “-1” if there is no match. We can name input j – “the index of the sub-array”, and because the original array has been separated into two sub-arrays based on the modulo of 2 (even or odd), the input j refers to (the index of the value in A) mod 2 (i.e., for the sub-array A1 j = 0, and for the A2 j = 1).

1. Write a pseudocode function called RecursiveFind, which is a recursive implementation of the function Find: it should take the same inputs as Find in addition to another integer i, and return the same integer as Find. HINT: the integer argument I should vary in the recursive function calls. [10 marks]

Function (RecursiveFind(key, B, M, j, i)

If(I >= M):

return -1

if(B[i] == key):

return j + 2 \* i

return RecursiveFind(key, B, M, j, i + 1)

1. The worst-case running time of a recursive implementation of Find for an array B of length M is T(M) = T(M-2) + c, where c is a constant. Very briefly explain why the Master Theorem is not relevant here for computing an expression of T(M) in terms of M. [5 marks]

The master theorem is used to compute the time for "divide and conquer" algorithms and based on the fact that the problem is split into smaller subproblems, these subproblems are solved recursively and, then subproblem solutions are combined to give a solution to the original problem. But the recursive find is more "decrease and conquer" algorithm: we don't split our problem and recombine solutions, but rather reduce it to a single smaller problem. Because there are no splitting/recombining solutions - we don't need the master theorem to calculate the computing time. We also can reason this out logically: RecursiveFind is a search algorithm that is based on comparisons (we don't have information about the original array being sorted), therefore if there no key value in the array (worst-case scenario), we need to look up all values inside the A.

Task 2: Consider the following pseudocode function that describes the R0 Search algorithm:

function R0(key,A1,A2,N)

index = Find(key,A1,ceiling(N/2),0)

if (index == -1):

return Find(key,A2,floor(N/2),1)

return index

In this pseudocode floor(x) and ceiling(x) are the mathematical functions that,

respectively, give the largest integer smaller than or equal to x and give the smallest integer

larger than or equal to x. For this algorithm address the following:

1. Identify, and describe very briefly in words, the best-case inputs and the worst-case

inputs. Recall that there are four inputs to R0. [8 marks]

2. An expression for both the worst-case and best-case running times (or execution time)

T(N), and describe the method by which you arrive at this expression. [8 marks]

3. The growth function of the worst-case and best-case running times T(N), i.e. a function

that does not include constants or low-order terms, e.g. if f(N) = 5N+2, then the growth

function is N. [5 marks]

4. The Theta notation for the worst-case and best-case running times T(N). In particular,

find a set of constants c1, c2 and m0 for which T(N) is ϴ(g(N)). [6 marks

|  |  |  |
| --- | --- | --- |
| **Function  - Line of code** | **Best-case input**  **(key, A1[key...], A2, N)**  For the best-case input, we assume that the first value of the A1 is equal to the key value, but the values themselves don’t matter | **Worst-case input**  **(key, A1[no key], A2[no key], N)**  For the worst-case input, we assume that there is no key value either in A1 nor in A2. |
| R0 – Line 1 (beside function calls) | Constant run time: C0 | Constant run time: C0 |
| Ceiling (assume constant runtime that doesn’t depend on argument) | Constant run time: C1 | Constant run time: C1 |
| Find – Line 1  Find – Line 2  Find – Line 3  Find – Line 4 | Because A1[0] = key the loop will iterate only once and return 0. Lines 1-3 will all have constant run times – C2, C3, C4, so the function “Find” has run time: C5 = C2 + C3 + C4 and returns the value 0 | Because there is no key value in the array A1, then line 1 will run Ceiling (N/2) + 1 times, line 2 – Ceiling (N/2), and line 3 will never fire up. Because the value of Ceiling(N/2) depends on N in a linear fashion, we can establish that the “Find” function has runtime: T(N) = C2\*N + C3 and returns -1 |
| R0 – Line 2  R0 – Line 3 (beside function calls) | Condition check has constant run time: C6, and because index = 0, line 3 will be skipped. | Constant run time: C4 |
| Floor (same as Ceiling) | Skipped (line 2 of R0 – false) | Constant run time: C5 |
| Find – Line 1  Find – Line 2  Find – Line 3  Find – Line 4 | Skipped (line 2 of R0 – false) | The same principle as the previous “Find” call, runtime T(N) = C6\*N + C7 and returns -1 |
| R0 – Line 4 | Constant run time: C7 | Skipped (line 3 of R0) |
| TOTAL RUN TIME | Best-case input has run time:  T(N) = C0 + C1 + C5 + C6 + C7 = C8. | The worst-case input run time:  T(N) = C0 + C1 + C2\*N + C3 + C4 + C5 + C6\*N + C7 = (C2 + C6)\*N + (C0+C1+C3+C4+C5+C7) = C8\*N + C9. |
| The growth function | 1 (Because run-time is constant) | N |
| The Theta notation | T(N) = С8 = Θ(N^0) = Θ(1)  c1 = {x∈ R+|x∈ (0;C8]}  c2 = {y∈ R+|y∈ [С8;+∞)}  m0 ∈ {z∈R+|z∈(0;+∞)}  (0; C8] ≤ С8 ≤ [C8;+∞) for all N > 0 | T(N) = С8 \* N + С9 = Θ(N)  c1 = {x∈ R+|x∈ (0;C8]}  c2 = {y∈ R+|y∈ [С8 + C9;+∞)}  m0 ∈ {z∈R+|z∈[1;+∞)}  (0; C8\*N] ≤ С8\*N + C9≤ [C8\*N + C9\*N; +∞) for all N ≥ 1 |

Task 3:Completethe following:

1.Write a pseudocode function R1(key, A, B, N)that takes a non-negative integer key, and arrays Aand Bof length Nas inputs: the function should return an index where the value keyis stored in the same location in both Aand B; it should return -1if the value cannot be found in either array; and it should return -2if there is an index jwhere A[i]is not equal to B[i].[8marks]

function R1(key, A, B, N)

for (0 ≤ i ≤ N): - we cycle through all values to check for the hardware value we don’t

if(A[i] =/= B[i]): search for the key value yet, because if there have been data loss after the

return -2 index that stores the key value in both arrays, we’ll miss the malfunction.

for (0 ≤ i ≤ N): - if data haven’t been altered, we can loop again to search the key value

if(A[i] == key): - we already confirmed that A[i] = B[i], so we can compare only one

return i

return -1

2.The Theta notationderivethe worst-caserunning time T(N)of the algorithm R1. Describe the steps taken in your reasoning about what is the worst-case input and the resulting worst-case running time.[6marks]

As we can see from the form of the algorithm: the wors-case input is two identical arrays that don't store the "key" value. In this case, the algorithm will have to cycle through 2 loops of length N and return -1 because data is ok, but the "key" value is missing. Therefore, if we translate “for (0 ≤ i ≤ N)” loop as {i = 0 ; if (i < N): … ; i = i + 1} and count each simple operation as taking 1 time unit, then lines 1 and 4 both will have runtime of 7N + 5. Lines 2 and 6 also equal in the terms of operations and have a runtime of 5N and the only return statement that will be executed (the last one) have a run-time of 1. The resulting runtime for the worst-case input of the R1 is T(N) = 24N + 5 and T(N) = Θ(N), because for n0 ∈ [5;+∞), c1 ∈(0;24], c2∈[25;+∞) => (0; 24N] ≤24N + 5≤ [25N;+∞) for all N ≥ 5.

Task 4:Completethe following:

1.Write a pseudocode function R1Hash(key, a, b, A, B, N)that takes non-negative integers key, aand bwhere aand bcome fromto the hashing function used to store indices in B; the arrays Aand Bof length Nand N2respectively are also inputs: the function should return an index iwhere the value keyis storedin array A; it should return -1if the value cannot be found; it should return -2if there was an error in the data storage, i.e. one of the values was altered unintentionally in at most one of the arrays.Recall that we are assuming that the hashing function avoids collisions in the hashed values for distinct values. [8marks]

function R1Hash(key, a, b, A, B, N)

Hash ← array[n^2]

for (0 ≤ i ≤ N^2):

Hash[i] = -1

for (0 ≤ i ≤ N):

h ← (a \* A[i] + b) % N^2

for (0 ≤ j ≤ N^2):

if (Hash[(h + j)%(N^2)] == -1):

Hash[(h + j)%(N^2)] = i

break

for (0 ≤ i ≤ N^2):

if (B[i] != Array[i]):

return -2

for (0 ≤ i ≤ N):

if (A[i] == key):

return i

return -1

2.Briefly explain your pseudocode, and how it returns the correct output in the three cases mentioned above.[6marks]

First of all, we check data integrity by creating a hash for array A using hash function variables ‘a’ and ‘b’, and comparing it to the hash B passed as an argument. We create and fill a new hash array (lines 1-3), and then for each value in A, we calculate the position of the index and change the value on the position ‘h’, or the next one until there is no collision (lines 4-9). After the correct hash array has been created, we can check each element of the passed hash array B, and if there was an error in data storage, it returns -2 (lines 10-12). If the data is OK, the algorithm will start to search for the key value in array A and return the index of the position ‘i’(lines 13-15) or the value ‘-1’ if there is no key value in array A (lines 16).