ALGORITHMS & DATA STRUCTURES 2

Mid-Term Assignment

Task 1:

1) The best-case input for the R0 Search algorithm would be when the value being searched for (the key) is found in the first element of either A1 or A2. The worst-case input would be when the value is not present in either array, or when it is present at the end of one of the arrays.
2) The worst-case running time for the R0 Search algorithm is O(n), since in the worst case the algorithm needs to search through both arrays A1 and A2. The best-case running time is O(1), since in the best case the algorithm only needs to search through one element of either array A1 or A2. These expressions were arrived at by considering the number of operations performed by the algorithm in the worst and best cases.
3) The growth function of the worst-case running time for the R0 Search algorithm is O(n). The growth function of the best-case running time is O(1). These expressions represent the asymptotic growth of the running time as the size of the input (N) increases. The growth function ignores constants and low-order terms, so the actual running time of the algorithm may be different in practice but will still exhibit the same asymptotic growth.

4) For the worst-case running time T(N), we can say that T(N) is Theta(N). This means that there exists constants c1, c2, and m0 such that T(N) is always between c1N and c2N for all N >= m0. For example, we could say that T(N) is Theta(N) with c1 = 1, c2 = 2, and m0 = 1. This means that for all N >= 1, the running time T(N) is always between N and 2N.

   For the best-case running time T(N), we can say that T(N) is Theta(1). This means that there exists constants c1, c2, and m0 such that T(N) is always between c1 and c2 for all N >= m0. For example, we could say that T(N) is Theta(1) with c1 = 1, c2 = 2, and m0 = 1. This means that for all N >= 1, the running time T(N) is always between 1 and 2.

Task 2:

1) The best-case and worst-case inputs for the CreateB function are the same because the running time of the function depends only on the structure of the input array A, not on the specific values of the elements in the array. In the worst-case, the array A is in random order, which requires the maximum number of recursive calls to calculate the sum. In the best-case, the array A is already sorted in ascending or descending order, which allows the Sum function to return the sum with the fewest number of recursive calls. In both cases, the running time of the

CreateB function is determined by the structure of the array A, not by the specific values of the elements in the array.

2) To derive the Theta notation for the worst-case and best-case running times of the CreateB function using the Master Theorem, we need to determine the asymptotic behavior of the recurrence relation that describes the running time of the Sum function.

   The Sum function has the following recurrence relation:
   T(N) = T(N/2) + O(1)

   To apply the Master Theorem, we need to determine the values of the constants a, b, and k such that the recurrence relation can be written in the form T(N) = a*T(N/b) + O(N^k). In this case, we can set a = 1, b = 2, and k = 0. This gives us the following recurrence relation:
   T(N) = T(N/2) + O(N^0)

   Applying the Master Theorem, we get the following result:
   T(N) = Theta(N^log_b(a)) = Theta(N^log_2(1)) = Theta(N^0) = Theta(1)

   This means that the worst-case running time of the Sum function is Theta(1), or constant time.

   For the best-case running time of the Sum function, we can again use the recurrence relation:
   T(N) = T(N/2) + O(1)

   Setting a = 1, b = 2, and k = 0, we get the following recurrence relation:
   T(N) = T(N/2) + O(N^0)

   Applying the Master Theorem, we get the following result:
   T(N) = Theta(N^log_b(a)) = Theta(N^log_2(1)) = Theta(N^0) = Theta(1)

   This means that the best-case running time of the Sum function is also Theta(1), or constant time.This means that the best-case running time of the Sum function is also Theta(1), or constant time.

Task 3:

1)

```
function R2(key, A, B, N)
if N is even
for i = 0 to N/2 - 1
if A[2i] + A[2i + 1] != B[i]
return -2
else if A[2i] == key or A[2i + 1] == key
return 2i
```

```
else
for i = 0 to N/2 - 1
if A[2i] + A[2i + 1] != B[i]
return -2
else if A[2i] == key or A[2i + 1] == key
return 2i
if A[N-1] == key
return N-1
return -1
```

2) In the worst case, the key is not found and no errors are detected. This means that the algorithm will iterate through all elements of A, checking each pair of elements against the corresponding element in B. The running time of the algorithm will therefore be Theta(N). This is because the number of iterations is directly proportional to the length of A, which is N.
3) Not all errors that can occur for array A will be detected in the method above because the method only checks for errors in pairs of elements in A. If an error occurs in a single element of A, it will not be detected by the method. This is because the method compares each pair of elements in A to the corresponding element in B, and an error in a single element of A will not affect the sum of that pair of elements.

Task 4:

1)

```
function R1(key, a, b, A, B, N)
for i = 0 to N-1
index = (a*A[i] + b) mod N
if A[i] != B[index]
return -2
else if A[i] == key
return i
return -1
```

2) To allow for repeated integers in array A, the method above can be adapted by using a hash table data structure instead of a simple array to store the values of array B. A hash table is a data structure that stores key-value pairs and uses a hash function to map keys to indices in an array, allowing for efficient insertion and search operations.

   In the case of our R1 function, we can use a hash table to store the values of array B. The hash function would still be used to map each element in array A to an index in the hash table, but in the case of collisions (when two or more elements in A hash to the same index in the hash table), the collision resolution strategy used by the hash table would determine how to handle the insertion of the key-value pairs into the hash table.

   For example, one collision resolution strategy is to use a linked list to store all the key-value pairs that hash to the same index. This way, when searching for a key in the hash table, we can iterate through the linked list at the hashed index to check if the key is present. If the key is found, we can return its value (which in our case would be the index of the key in array A). If the key is not found in the hash table, we can return -1 as before. If an error is detected (when the value stored at a given index in the hash table does not match the corresponding element in array A), we can return -2 as before.

Task 5:

1)

```
function MySearch(key, A, N)
x = hash(key, N)
y = hash(key * 3 + 1, N)
for i in 0 to N-1
if A[x] == key or A[y] == key:
return i
x = (x + 1) mod N
y = (y + 1) mod N
return -1
```

2) This function uses two hash functions to store the data in two different locations in the array A. The first hash function is simply (key mod N), and the second is (key * 3 + 1 mod N). These two hash functions should distribute the data evenly throughout the array, and minimize the chances of a collision. The function then searches both hashed locations for the key, and if it is found, returns the index in the original array. If the key is not found after searching both hashed locations, the function returns -1.
3) The worst-case input for my search algorithm would be when the value of key is not present in the original array and the data storage is error-free. In this case, the algorithm would have to search through the entire array and all of its copies without finding a match, resulting in a longer execution time.

   On the other hand, the best-case input would be when the value of key is present in the first element of the original array and the data storage is error-free. In this case, the algorithm would immediately find a match and return the index, resulting in a shorter execution time.
4) To derive the worst-case running time for the search algorithm, we need to consider the length of the original array, the number of copies of the array, and the number of elements that must be checked in each copy before finding a match or determining that there is an error in the data storage.

   Let's assume that the original array has a length of N, and we have M copies of the array. The worst-case running time can be expressed as T(N, M) = N * M. This means that the algorithm will take N * M steps to search through all of the elements in the original array and all of its copies.

   To derive the best-case running time, we can again consider the length of the original array and the number of copies. However, in this case, we only need to consider the first element of the original array and its corresponding element in the first copy of the array. The best-case running time can be expressed as T(N, M) = 1. This means that the algorithm will only take one step to find a match and return the index.

5) To determine the Theta notation for worst-cast inputs, we need to find a set of constants c1, c2, and m0 such that the worst-case running time T(N) is within the range c1 * g(N) <= T(N) <= c2 * g(N) for all N >= m0.

   For example, if the worst-case running time of the algorithm is T(N) = 5N^2 + 3N + 2, and we can find constants c1, c2, and m0 such that 5N^2 <= 5N^2 + 3N + 2 <= 10N^2 for all N >= 10, then we can say that the Theta notation for the worst-case running time is Theta(N^2).

   To express the best-case running time using Theta notation, we would follow the same process and identify the growth function of the best-case running time. For example, if the best-case running time is O(1) (constant time), the Theta notation for the best-case running time would be Theta(1).