Task 1:

1) The best-case inputs for the R0 function are when the value being searched for is present in the first element of A1. In this case, the function will return 0 after just one comparison. The worst-case inputs are when the value being searched for is not present in either A1 or A2, in which case the function will have to compare every element of both arrays before returning -1.
2) The worst-case running time of the R0 function is when it has to compare every element of both A1 and A2 before returning -1. Since A1 and A2 have lengths ceiling(N/2) and floor(N/2) respectively, the worst-case running time is T(N) = ceiling(N/2) + floor(N/2) = (N/2 + 1) + (N/2) = N. The best-case running time is when the function returns after the first comparison, which occurs when the value being searched for is present in the first element of A1. In this case, the running time is T(N) = 1.
3) The growth function of the worst-case running time T(N) is N, since the running time is directly proportional to the size of the input. The growth function of the best-case running time is 1, since the running time is constant and does not depend on the size of the input.
4) The Theta notation for the worst-case running time T(N) is Theta(N), since T(N) is proportional to N and there exist constants c1 and c2 such that c1 * N <= T(N) <= c2 * N for all values of N greater than or equal to some constant m0. For example, we can set c1 = 1, c2 = 2, and m0 = 1, so that for all N >= 1, 1 * N <= T(N) <= 2 * N. The Theta notation for the best-case running time T(N) is

Theta(1), since T(N) is constant and does not depend on the size of the input.

Task 2:

1) The best-case inputs for the CreateB function are when the array A is already sorted in ascending order. In this case, the function will have to make the fewest possible number of recursive calls to the Sum function in order to calculate the sum of the elements in A. The worst-case inputs are when the array A is already sorted in descending order. In this case, the function will also have to make the fewest possible number of recursive calls to the Sum function, since the array is already sorted. Therefore, the best-case and worst-case inputs for CreateB are the same.
2) To derive the Theta notation for the running time of the CreateB function, we can use the Master Theorem. The function Sum makes two recursive calls to itself, and the size of the problem being solved decreases by a factor of 2 each time. Therefore, we can express the running time of Sum as T(N) = 2T(N/2) + O(1). Using the Master Theorem, we can then determine the Theta notation for the running time of CreateB as follows:

If a = 2, b = 2, and f(N) = O(1), then log_b(a) = log_2(2) = 1, which means that T(N) is in Theta(N^1 log N).

If a = 2, b = 2, and f(N) = O(N^(log_b(a) - 1)), then log_b(a) - 1 = 1 - 1 = 0, which means that T(N) is in Theta(N^0).

If a = 2, b = 2, and f(N) = O(N^(log_b(a))), then log_b(a) = 1, which means that T(N) is in Theta(N^1).

If a = 2, b = 2, and f(N) = O(N^(log_b(a) + 1)), then log_b(a) + 1 = 1 + 1 = 2, which means that T(N) is in Theta(N^2).

Since f(N) = O(1) for the function Sum, the running time of CreateB is in Theta(N log N). This means that the worst-case and best-case running times of CreateB are both in Theta(N log N).

Task 3:

1) Here is a pseudocode function R2(key, A, B, N) that takes a non-negative integer key, and arrays A and B, as well as the length of A, N:

```
function R2(key, A, B, N)
  for 0 <= i < N
    if A[i] == key:
      return i
  if sum(A) != sum(B):
    return -2
  return -1
```

This function performs a linear search through A to check if the key is present in A. If it is, the function returns the index at which the key was found. If the key is not found in A, the function checks if the sums of the elements in A and B are equal. If they are not, the function returns -2 to indicate that an error has been detected. If the key is not found in A and there is no error detected, the function returns -1 to indicate that the key was not found.

2) The worst-case running time of the algorithm R2 occurs when the key is not present in A and there is no error detected. In this case, the algorithm will have to perform a linear search through A, which will take O(N) time. Therefore, the worst-case running time of the algorithm R2 is T(N) = O(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 by comparing the sums of the elements in A and B. There may be other errors that can occur in A that are not reflected in the sum of the elements in A and B, and therefore will not be detected by this method. For example, if two elements in A are swapped, this will not change the sum of the elements in A and B, but it will still be an error in A.

Task 4:

1) Here is a pseudocode function R1(key, a, b, A, B, N) that takes non-negative integers key, a and b where a and b come from to the hashing function used to store indices in B; the arrays A and B of length N are also inputs:

```
function R1(key, a, b, A, B, N)
  # Hash the value using the given function
  hash_index = (a*key + b) mod N
  # Check if the hashed value is present in B
  if B[hash_index] == key:
    return A.index(key) # Return index in A
  # Check if the key is present in A
  if key in A:
    return -2 # Error detected
  return -1 # Key not found
```

This function will hash the value using the given hashing function and check if it is present in B at the index determined by the hashing function. If it is, the function will return the index of the value in A. If the hashed value is not present in B, the function will check if the key is present in A. If it is, the function will return -2 to indicate that an error has been detected. If the key is not present in either A or B, the function will return -1 to indicate that the key was not found.

2) To allow for repeated integers in array A, the method above could be adapted by using a hash table to store the indices of the repeated integers in A. In this case, when the key is hashed using the given function, the function could check if the hashed value is present in the hash table. If it is, the function could retrieve the indices of all

the occurrences of the key in A from the hash table and return them. If the hashed value is not present in the hash table, the function could check if the key is present in A using a linear search. If it is, the function could store the index of the key in the hash table and return it. If the key is not present in either the hash table or A, the function could return -1 to indicate that the key was not found. This would allow the function to efficiently handle repeated integers in array A while still being able to detect errors in the data storage.

Task 5:

1) One potential setting for storing and searching data redundantly could be as follows:

We start with an array A of length N storing non-negative integers. We then create two new arrays, A1 and A2, of length N/2. The array A1 stores the values A[i] where i is an even number, and the array A2 stores the values A[j] where j is an odd number. In addition, we create a third array C of length N/2 that stores the sum of the corresponding elements in A1 and A2.

For example, if A = [5, 3, 8, 6, 1, 9], A1 would be [5, 8, 1] and A2 would be [3, 6, 9]. The array C would be [8, 14, 10].

To search for a particular value in this setting, we could use the following pseudocode function:

```
function search(key, A1, A2, C, N)
  # Check if the key is present in A1
```

```
for i in 0 to N/2:
  if A1[i] == key:
    return 2*i
# Check if the key is present in A2
for i in 0 to N/2:
  if A2[i] == key:
    return 2*i + 1
# Check if the key is present in C
for i in 0 to N/2:
  if C[i] == key:
    return -2
```

2) The pseudocode function I provided, search(key, A1, A2, C, N), works by iterating through each array and checking if the key is present in each one. If the key is found in any of the arrays, the function returns the appropriate index or error value. If the key is not found in any of the arrays, the function returns -1 to indicate that the key was not found.

3) The worst-case inputs for the search algorithm would be when the key is not present in any of the arrays. This would require the function to iterate through all the elements of each array, which would take the longest amount of time. The best-case inputs for the search algorithm would be when the key is present in the first element of one of the arrays. This would require the function to only iterate through a single element, which would take the least amount of time.

4) The worst-case running time for the search algorithm is O(3N/2), as the function must iterate through all the elements of each array in the worst case. The best-case running time is O(1), as the function only needs to check a single element in the best case.

5) The Theta notation for the worst-case running time is Θ(N), as the function iterates through N elements in the worst case. The Theta notation for the best-case running time is Θ(1), as the function only needs to check a single element in the best case.