## Task 1

1. The best case is the array with a total length (N) is even and the item (key) being searched is the first element of the even indexed array (A1) on the first loop.
   An example is an array with A = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] with A1 = [1, 3, 5, 7, 9], A2 = [2, 4, 6, 8, 10], key = 1, N = 10
   The algorithm finds 1 on the first iteration of A1 elements

   The worst case is when the item (key) being sought is in the last element of the odd indexed array (A2) or the key is not present in both even and odd indexed arrays (A1 and A2). A = [1, 2, 3, 4, 5, 6, 7, 8, 9], A1 = [1, 3, 5, 7], A2 = [2, 4, 6, 8], key = 8, N = 9

2. For best case
   T(N) = C + C + C + C = 4C
   Because there's only one initialization, 2 comparisons and one return only taking constant times
   For worst case
   T(N) = line 2 (C + (ceil(N/2) + 1) + ceil(N/2)) + line 3 and 4 (ceil(N/2) + C) + line 5 (C + (floor(N/2) + 1) + floor(N/2)) + line 6 and 7 (floor(N/2) + C) + line 8 (C)
   T(N) = 5C + 3ceil(N/2) + 3floor(N/2) + 2
   Since ceiling and floor are constant
   T(N) = 5C + 3(N/2) + 3(N/2) + 2 = 5C + 6(N/2) + 2

3. In best case; its 1(constant time)
   In worst case it's N

4. Using the expression: $c1 * g(N) \leq T(N) \leq c2 * g(N)$
   For best case
   Choose g(N) = 1 , c1 = 1, c2 = 3, m0 = 1
   $\Theta(g(N)) = 1 \leq f(1) \leq 3$
   For worst case
   Choose g(N) = N, c1 = 1, c2 = 2, m0=1
   $\Theta(g(N)) = N \leq f(N) \leq 2N$

## Task 2

1. The best case is when the input array A of N length is sorted in ascending order and in the worst situation, A is in descending order and since the CreateB always calls Sum with the same inputs, best-case and worst-case inputs for CreateB are the same.
2. Sum has a running time of T(N/2) in the worst-case and a running time of T(1) in the best-case.
   ∴ CreateB recurrence relation is: T(N) = a * T(N/b) + f(N)

Where a is the number of recursive calls(1), b is the input size (2), and f(N) is the non-recursive work time (1).
$f(N) = n^{\log}b^{a} = 1$. Therefore, $T(N) = \Theta(\log n)$

## Task 3

1.

The pseudocode for R2 is as follows:

```
function R2(key, A, B, N):
B = CreateB(B, length(B)) //This creates a new array B with the first element as the sum of
// all of the values in B
if B[0] != Sum(A, 0, N-1): //Sum(A, 0, N-1) is the sum of all values in A
  return -2
A1 = new Array of length ceil(N/2) //ceil(N/2) is used because if N is odd, the last element of
// A1 will be the last element of A
A2 = new Array of length floor(N/2) //floor(N/2) is used because if N is odd, the last element
//of A2 will be the second last element of A
for 0 <= i < N:
  if i mod 2 = 0: //if i is even
    A1[i/2] = A[i] //i/2 will always be an integer that's an index of A1
 else:
   A2[(i-1)/2] = A[i] //i-1 will always be an even number, and (i-1)/2 will always be an integer
// that's an index of A2
 end for
 return R0(key, A1, A2, N) //R0 is the function that finds the index of a key in an array
end function
```

2. The worst-case running time of R2 it's $\Theta(\log N)$ because the worst-case running time of R0 is $\Theta(N)$ and the worst-case running time of CreateB is $\Theta(N)$ and the worst-case running time of Sum is $\Theta(N)$
   Then worst-case running time of R2 is $(\log N) + O(N) + O(N) + O(N) = O(N)$

   The worst case input for R2 is an array A in descending, where N is a power of 2, and the key is not in A.

3. This is because only sum is used for error checking. An error that results in a change in multiple elements and/or their order in A is not detected.

## Task 4

1.

The pseudocode for R1 is shown below:

```
function R1(key, a, b, A, B, N)
j = (a*key + b) mod N
if B[j] = key:
  for 0 <= i < N: // This finds the index i where the value key is stored in array A
    if A[i] = key:
      return i
  end for
else if ISNULL(B[j]): //if the value does not exist in B
  return -1
else: //if B[j] has another value apart from key it means that there's a mismatch which is an error
//in the data storage
  return -2
end function
```

2. It can be adapted by using an array that keeps counts of elements in A. When a key is found the count is checked, if it's repeated, iteration continues. Another way is using linear probing to extend B and return the indices instead.

## Task 5

1.

The pseudocode for the search algorithm is shown below:

```
function SearchDataWithSeparateChaining(key, a, b, A, B, N)
  j = (a * key + b) % N
// Detecting data storage error by checking if all the elements in A are in correct position in B
  for i = 0 to N - 1
    j = (a * A[i] + b) % N
    for k = 0 to length(B[j]) - 1 //This checks if the value A[i] is in the array B[j]
      if B[j][k] == A[i]
        break
      else if k == length(B[j]) - 1
        return -2
    end for
  end for
  for i = 0 to length(B[j]) - 1 //Search for key in B[j]
    if B[j][i] == key
      for k = 0 to N - 1
        if A[k] == key //find the index k where the key is stored in A
          return k
      end for
    end if
  end for
  return -1
end function
```

Additionally, This is the pseudocode that creates a hash table with separate chaining

```
function createArrayBSeparateChaining(key, a, b, A, N)
  B = new array of length N
  for i = 0 to N - 1
    B[i] = new array
  end for
  for i = 0 to N - 1
    j = (a * A[i] + b) % N
    B[j].push(A[i]) //This adds the value A[i] to the end of the array B[j]. It takes O(1) time.
  end for
  return B
end function
```

2. The pseudocode works because works by first calculating the index j of the array B where the key is expected to be stored using the hashing function (a * key + b) % N. It then checks if all the elements in array A, are stored in the correct position in array B by iterating through the elements in A and using the same hashing function to calculate the expected index in B for each element. If an element in A is not found in its expected position in B, the function returns -2, indicating an error in the data storage.

If all the elements in A are found in their expected positions in B, the function proceeds to search for the key in B[j] by iterating through the elements in B[j] and checking if any of them are equal to the key. If the key is found in B[j], the function then iterates through the elements in A to find the index k where the key is stored in A, and returns k. If the key is not found in B[j], the function returns -1.
The pseudocode was also implemented in a real programming language.

3. The worst-case input for the algorithm is when the value of the key is not present in B[j] and all the elements in B[j] need to be checked before returning -1. The best-case input is when the value of the key is present in the first position of B[j] and is immediately returned.

4. The algorithm's worst-case execution time is O(N^2) because, in the worst-case situation, the for loop iterating through A will need to check every element in B[j] for every element in A. The algorithm's best-case execution time is O(1), as in the best-case situation, the value of the key is immediately returned.

5. The worst-case running time is represented by Theta(N^2), whereas the best-case running time is represented by Theta (1).