Algorithms Task: 4. Birthday Cake Candles

Implementation

Algorithm #1 Non-Recursive

```
#include <stdio.h>

int birthdayCakeCandles(int candles[] ,int n){
   int max = candles[0], counter = 0;
   for(int j = 1; j < n ; j++) {
      if (max < candles[j])
         max = candles[j];
   }
   for(int i = 0; i < n ; i++){
      if(max == candles[i])
      counter++;
   }

   printf("\nThe Number is: %d", counter); }

int main() {
   int n;
   printf("How many candles are there? ");
   scanf("%d", &n); int candles[n];
   printf("Enter the candles numbers:\n");
   for(int i = 0; i <= n-1; i++)
      scanf("%d", &candles[i]);
   birthdayCakeCandles(candles, n);
   return 0;
}
```

Algorithm #2 Non-Recursive

```
#include <stdio.h>

int birthdayCakeCandles(int candles[] ,int n){
   int max = candles[0], counter = 1;
   for(int j = 1; j < n ; j++) {
      if (max < candles[j]) {
         max = candles[j];
         counter = 1;
```

```
        } else if (max == candles[j]){
            counter++;
        }
    }
    printf("\nThe Number is: %d", counter); }

int main() {
    int n;
    printf("How many candles are there? ");
    scanf("%d", &n); int candles[n];
    printf("Enter the candles numbers:\n");
    for(int i = 0; i <= n-1; i++)
        scanf("%d", &candles[i]);
    birthdayCakeCandles(candles, n);
    return 0;
}
```

Algorithm #3 Recursive

```
int birthdayCakeCandles(int arr[], int n, int max, int cnt,int tempN){
    if(n == 0){
        if(tempN == 0)
            return cnt;
        else{
            if(arr[tempN-1] == max) ++cnt;
            birthdayCakeCandles(arr,n,max,cnt,tempN-1);
        }
    }
    else{
        if(arr[n-1] > max) max = arr[n-1];
        birthdayCakeCandles(arr,n-1,max,cnt,tempN);
    }
}


void solve(){
    cout << "How many candles are there ? ";
    int n; cin >> n;
    int arr[n];
    cout << "Enter the candles numbers : ";
    for (int i = 0; i < n; i++)
    {
```

```
        cin >> arr[i];
    }
    int x = birthdayCakeCandles(arr,n,INT_MIN,0,n);
    cout << "the count of the largest candles is : " <<x ;
}

int main(){
    solve();
    return 0;
}
```

Pseudocode

Algorithm #1 Non-Recursive Pseudocode

```
birthdayCakeCandles(candles[0..n-1])

max := candles[1]
counter := 0
for j := 1 to n do
    if max < candles[j]
        max := candles[j]
for  i := 0 to n do
    if max = candles[i]
        counter := counter +1

return counter
```

Algorithm #2 Non-Recursive Pseudocode

```
birthdayCakeCandles(candles[0..n-1])

max := candles[0]
counter := 1
for j := 1 to n do
    if max < candles[j]
        max := candles[j]
        counter <- 1
    else if max = candles[j]
```

```
        counter := counter + 1

return counter
```

Algorithm #3 Recursive Pseudocode

```
birthdayCakeCandles(candles, n, max, count, temp N)

if n = 0
   if tempN = 0
      return count
   else
       if(candles[tempN-1] = max)
          count := count = 1
          birthdayCakeCandles(candles,n,max,count,tempN-1)
else
   if candles[n-1] > max
      max := candles[n-1]
      birthdayCakeCandles(candles,n-1,max,count,tempN)
```

Anslysis & Complexity

Algorithm #1 Non-Recursive Analysis:

Basic Operation:

```
for(int j = 1; j < n ; j++) {
        if (max < candles[j])
            max = candles[j];
or
 for(int i = 0; i < n ; i++){
        if(max == candles[i])
        counter++;
```

The function goes over all elements in the array to find out the biggest element then goes over it again to count them.

The Comparison (if (max < candles[j]) or if(max == candles[i]))
will be repeated n number of times therefore the Worst, Best and Average time is:
O(n) (*O(2n) -> O(n)*)

Algorithm #2 Non-Recursive Analysis:

Basic Operation:

```
for(int j = 1; j < n ; j++) {
        if (max < candles[j]) {
            max = candles[j];
            counter = 1;
        } else if (max == candles[j]){
            counter++;
        }
```

The function goes over the elements in the array starting from the second one having the first element as the max then checks if there is an element greater than the previous and increments the count every time it sees the max number again or change the count back to 1 if a newer and greater element was discovered.

The Comparison (if (max < candles[j]) and else if (max == candles[j])) will be both repeated n number of times therefore The Best, Worst and Average time is: **O(n)**

Algorithm #2 Recursive Analysis:

Basic Operation:

Comparison

Best Time Complexity to Worst:
1. Algorithm #2: O(n) where n = number of elements in array
2. Algorithm #1: *O(2n)* -> *O(n)* where n = number of elements in array
3. Algorithm #3: