CPSC 335

Project 2

WeCSharp

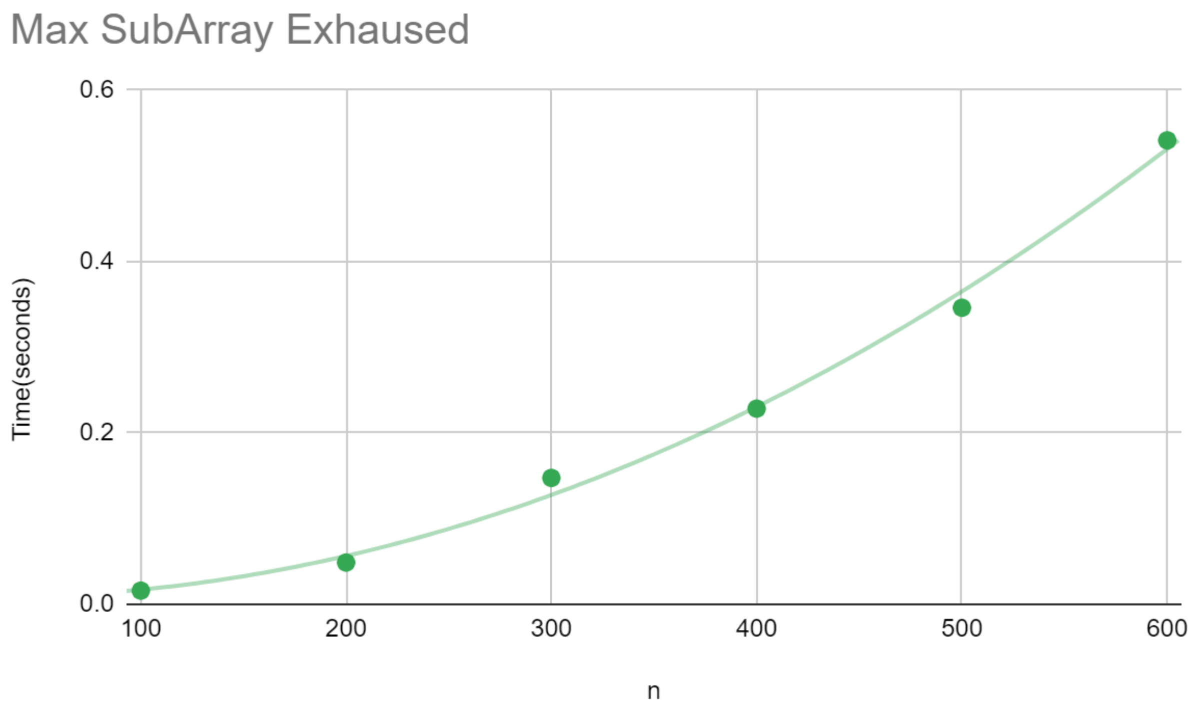
Casey Thatsanaphonh
cthatsanaphonh@csu.fullerton.edu

Jennifer Felton
jfelton@csu.fullerton.edu

Maximum Subarray Exhausted Search:

```
maximum_subarray_exh(V):
    b = 0, e = 1
    for i from 0 to n-1:
        for j from i+1 to n:
            if sum(V[i:j]) > sum(V[b:e]):
                b = i
                e = j
    return (b, e)
```

Hypothesis: Based on the pseudo code provided for the function we can see there is a nested for loop that has a time complexity of O(n^2) and the provided sum function which has a time complexity of O(n) we can see the suspected trend of the graph should be polynomial. The function will have a suspected time complexity of O(n^3).



n	Time(seconds)
100	0.0159078
200	0.048696
300	0.147471
400	0.228339
500	0.346083
600	0.541489

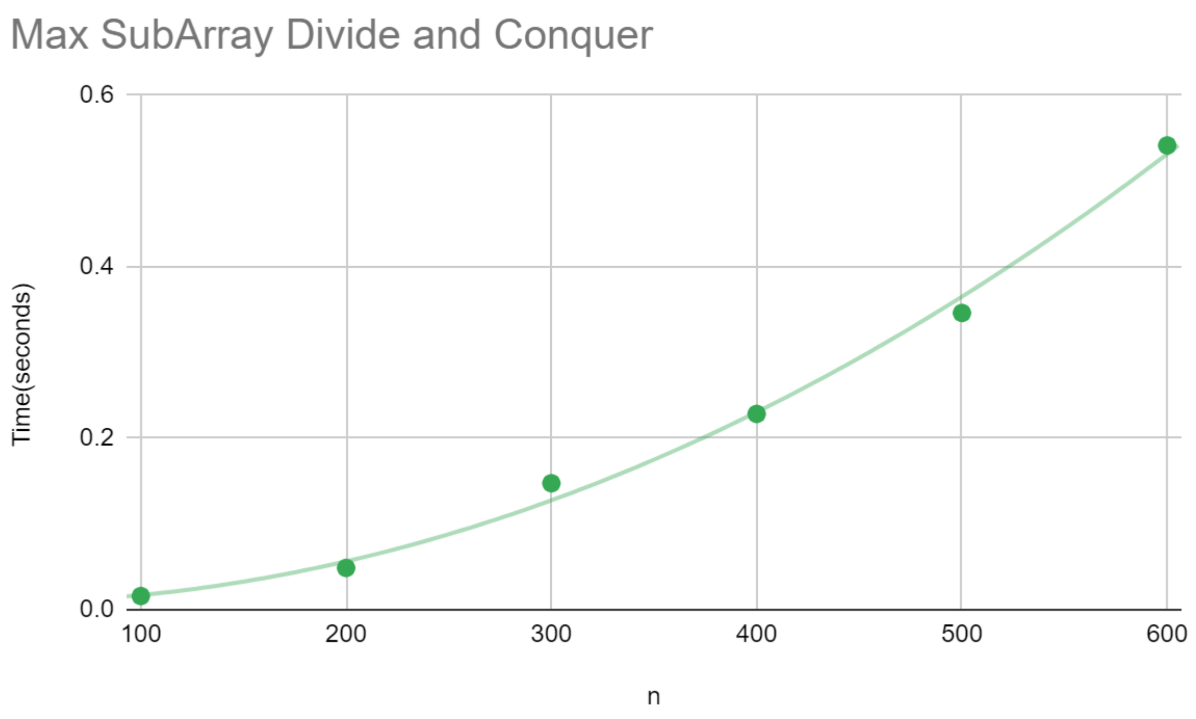
Based on the data collected and inputted into the graph we can see the best fit line for the graph is polynomial. The data collected is consistent with the original hypothesis about the time complexity of the graph. The original hypothesis for the maximum subarray exhausted search function was it would have a time complexity of O(n^3) and this is of polynomial time which is also reflected on the graph with the best fit line.

Max SubArray Divide and Conquer:

```
maximum_subarray_dbh(V):
    return maximum_subarray_recurse(V, 0, n-1)

maximum_subarray_recurse(V, low, high):
    if low == high:
        return (low, low + 1)
    middle = (low + high) / 2
    entirely_left = maximum_subarray_recurse(V, low, middle)
    entirely_right = maximum_subarray_recurse(V, middle + 1, high)
    crossing = maximum_subarray_crossing(V, low, middle, high)
    return (whichever of entirely_left,
                         entirely_right, and
                         crossing
            have the greatest sum)
```

Hypothesis: Based on the pseudo code the maximum_subarray_dbh function should use a helper function to recursively iterate through the vector and split the vector into two halves continuously until the high and low are the same. This approach is $T(n) = 2T(n/2)+n$ and using master theorem the time complexity of the maximum_sumarray_dbh function will have a time complexity of O(nlogn).



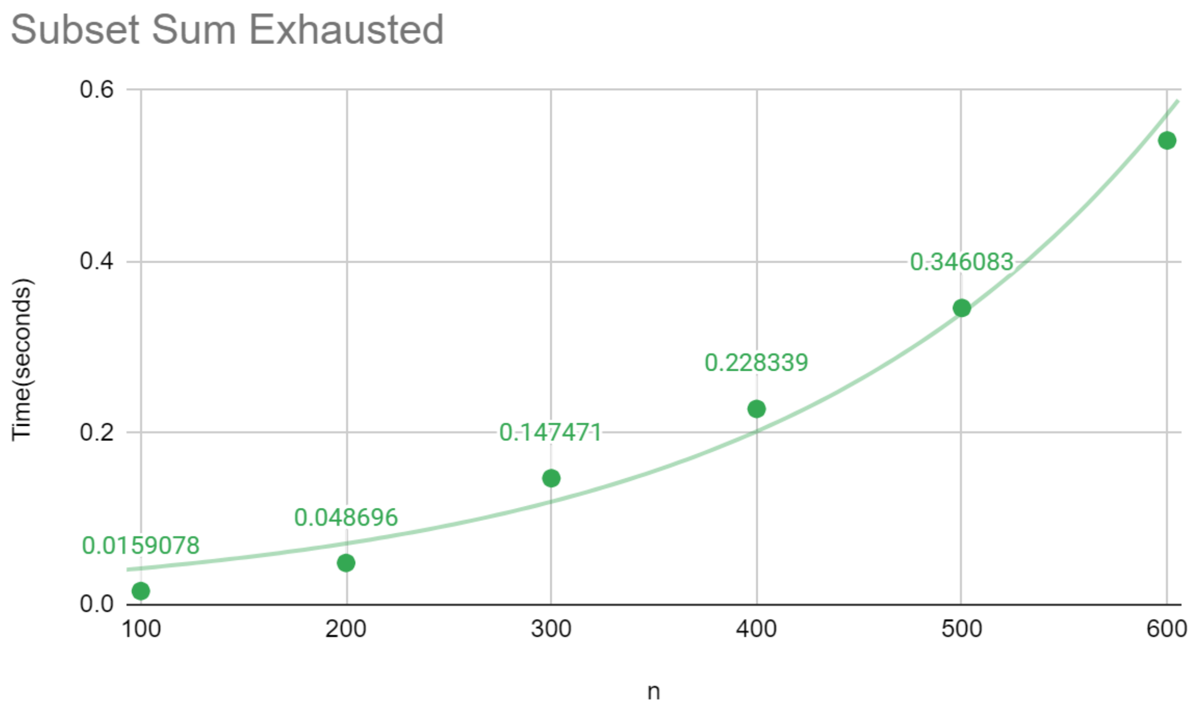
n	Time(seconds)
100	0.015078
200	0.04896
300	0.140771
400	0.2233399
500	0.3600983
600	0.540149

Based on the data collected we can see that the function does not have an O(nlogn) time complexity. This is not consistent with the previously stated hypothesis. The approach taken was the greedy algorithm approach and the outcome was much different than expected had the recursive approach be used.

Subset sum exhausted Search:

```
subset_sum_exh(U, t):
  n = |U|
  for bits from 0 to (2n -1):
     candidate = []
     for j from 0 to n-1:
        if ((bits >> j) & 1) == 1:
           candidate.add_back(U[j])
     if |candidate| > 0 and sum(candidate) == t:
        return candidate
  return None
```

Hypotheseis: Based on the pseudo code we can see the initial loop will iterate 2n-1 times while the nested loop inside will iterate n-1 times. We will be utilizing the bitwise operations and the suspected time complexity for the function will be exponentially growing with a time complexity of O(2n^2)



n	Time (seconds)
10	0.000131495
12	0.00064315
14	0.00304016
16	0.013637
18	0.0626196
20	0.268023

Based on the data collected and inputted into the graph we can see the subset_sum_exh function has an exponentially growing time complexity. With the provided information this function should have O(n*2^n) time complexity. With the points plotted on the graph we can clearly see with the best fit line that the time efficiency data is consistent. The hypothesis created states the suspected time complexity will be exponentially growing and we can see with the data we collected that this prediction is consistent.

1. There is a noticeable difference in performance between the three algorithms, the first thing to note is the subset sum function could not accept n values above 20 or the function would fail the tests and ran significantly slower than the other two functions. The fastest algorithm is the decrease by half subarray function and exhausted search version and this was due to not utilizing the recursive approach.
2. The empirical analysis was consistent with the predicted big-O efficiency class for each algorithm and we can determine this when the data is plotted on the graph and we use the best fit line which accurately reflects each algorithm after 6 tests completed.
3. The evidence is consistent with hypothesis 1 because we suspected the time complexity to be O(n^3) and when we graphed our data we can see the graph to be polynomial proving to be consistent to what was originally proposed.
4. The evidence is not consistent with hypothesis 2 because we suspected the time complexity of the function to be O(nlogn) but the approach ended up with a polynomial time.
5. The evidence is consistent with hypothesis 3 because we suspected the time complexity to be O(n*2^n) and when we graphed our data we can see the graph to be exponentially growing; proving the hypothesis to be consistent with our evidence.