Programming Assignment #2 - CS325

Joshua Villwock Jaron Thatcher Ryan Phillips

February 25, 2014

Introduction

In this writeup, we will both assess time complexity and prove three algorithms designed to solve the Maximum Subarray problem.

According to Wikipedia:

'In computer science, the maximum subarray problem is the task of finding the contiguous subarray within a one-dimensional array of numbers (containing at least one positive number) which has the largest sum. For example, for the sequence of values 2, 1, 3, 4, 1, 2, 1, 5, 4; the contiguous subarray with the largest sum is 4, 1, 2, 1, with sum 6.'

Pseudocode

Brute Force:

```
def brute_force(x):
  max_sum = 0
  l = len(x)
  for i in xrange(l):
    new_sum = 0
    for j in range(i,l):
      new_sum += x[j]
      if new_sum > max_sum: max_sum = new_sum
  return max_sum
```

Divide and Conquer:

```
def div_and_conq(x):
  max_sum = 0
  def inner(x,max_sum):

    if (len(x) <= 1): # base case
      if sum(x) > max_sum:
        max_sum = sum(x)
      return max_sum

    mid = int(len(x)/2) # split into two halves
    l = x[:mid]
    r = x[mid:]

    max_sum = max(max_sum, sum(l)) # is left half max?
    max_sum = max(max_sum, sum(r)) # is right half max?

    # does max consists of suffix+prefix?
    suffix_sum = 0
    max_suffix_sum = 0
    for i in range(mid-1,-1,-1):
      suffix_sum += x[i]
      max_suffix_sum = max(max_suffix_sum, suffix_sum)
    prefix_sum = 0
```

```
        max_prefix_sum = 0
        for i in range(mid,len(x)):
            prefix_sum += x[i]
            max_prefix_sum = max(max_prefix_sum, prefix_sum)
        max_sum = max(max_sum, max_suffix_sum + max_prefix_sum)

        # recursive calls
        ret = inner(l,max_sum)
        if (ret != None):
            max_sum = max(ret,max_sum)
        ret = inner(r,max_sum)
        if (ret != None):
            max_sum = max(ret,max_sum)

        return max_sum # end of inner function

    return inner(x,max_sum)
```

Dynamic Programming:

```
def dynamic_prog(x):
    this_sub_arr_sum = 0
    max_sum = 0
    for i in x:
        if this_sub_arr_sum + i > 0:
            this_sub_arr_sum = this_sub_arr_sum + i
        else:
            this_sub_arr_sum = 0
        if this_sub_arr_sum > max_sum:
            max_sum = this_sub_arr_sum

    return max_sum
```

Correctness Proofs

Divide and Conquer:

Dynamic Programming:

Asymptotic Analysis of Run Time

Brute Force:

Divide and Conquer:

Dynamic Programming:

Empirical Testing of Correctness

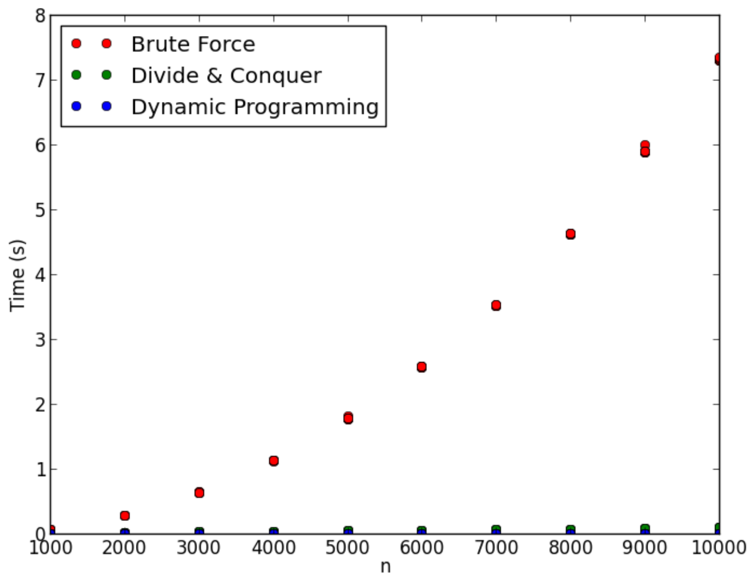
Correctness of the three algorithms was verified using the provided file of test cases verify_2.txt. See the function 'test_correctness' in main.py for details.

Here is the output from the final text input file (name.txt):

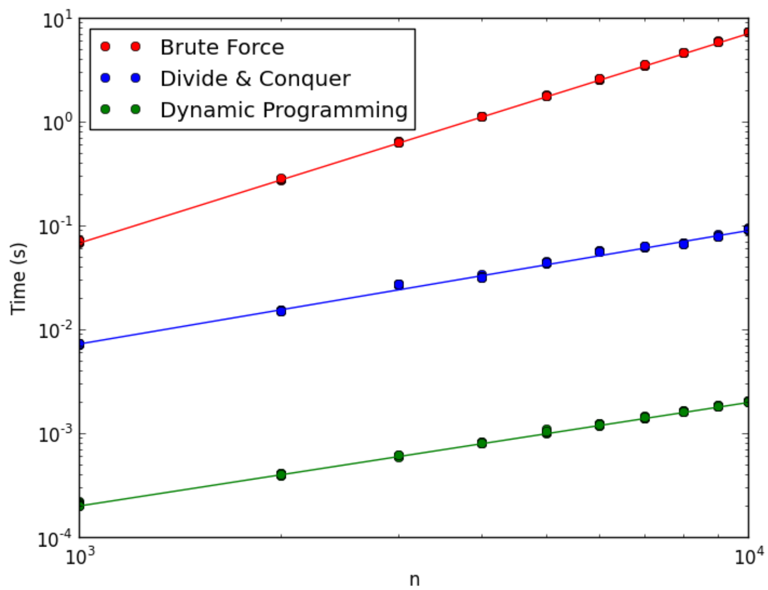
- OUTPUT HERE (TODO)

Empirical Analysis of Run Time

Linear Plot:



Log-log Plot:



Slope of lines in log-log plot:

The equation for the best fit line on the log-log plot (calculated using numpy.polyfit()) has the following form:

$f(n) = e^{\text{y-intercept}} * n^{\text{slope}}$

Brute Force:

- slope: 2.0177140011

Divide & Conquer:

- slope: 1.09279503259

Dynamic Programming:

- slope: .996887276493

Performance Comparison

'In your report, present and compare the empirical run time results of the three different algorithms. Provide a discussion of the comparative benefits and drawbacks of different algorithms.'