Programming Assignment #1 - CS325

Joshua Villwock Jaron Thatcher Ryan Phillips

January 29, 2014

Pseudocode

Brute Force:

```
BruteForce(arr)
  count = 0
  for i in 0 to arr.length
    for j in i to arr.length
        if arr[i] > arr[j]:
        count++
  return count
```

Naive Divide and Conquer:

```
NaiveDivideAndConquer(arr)
  count = 0
  if len(arr) < 2:
    return count
  middle = length(list_in)/2
  left = arr[:middle] // slice off half of the array
  right = arr[middle:]
  // count inversions between left and right halves
  for i in range (0,len(left)):
    for j in range (0,len(right)):
      if left[i] > right[j]:
          count++
  // and count internal inversions recursively
  count += NaiveDivideAndConquer(left)
  count += NaiveDivideAndConquer(right)
  return count
```

Merge and Count:

```
MergeAndCount(arr,0)
  results = []
  // base case
  if len(x) < 2:
    return x, count
  middle =len(x)/2
  // recursive calls
  left, count = MergeAndCount(x[:middle],count)
  right, count = MergeAndCount(x[middle:],count)
  i, j = 0, 0
  while i < length(left) and j < length(right):
    if left[i] > right[j]:
      results.append(right[j])
      count += length(left) - i
      j++
    else:
      results.append(left[i])
      i++
    results += left[i:]
    results += right[j:]
    return results, count
```

Correctness Proof

Asymptotic Analysis of Run Time

Brute Force: It has two for loops of size n duh

Naive Divide and Conquer: T(n) = this class is difficult

Merge and Count: T(n) = T(n/2) + O(n) T(n) = T(n/2) + cn //1st recursion -Telescoping -[T(n/4) + cn] + cn = T(n/4) + 2cn //2 -[T(n/8) + cn] + 2cn = T(n/8) = 3cn //3

-General Pattern: T(n/2$^n) + ncn$

When T(n) = T(n/(2$^n) + ncn - Youhavetwosubproblemsofsizen - Pluslineartimecombination$

-AKA: O(n log n)

Testing

The first test for correctness was performed using the provided file "verify.txt". It was assumed that the last value of each row was the expected number of inversions, so all 3 algorithms were run on each row of values (excluding the last), and this was compared to the expected value. This can be performed via: "test_correctness1("verify.txt")".

The second test for correctness used the second provided file "test_in.txt". Since no expected values were given, the results were just printed out. All 3 algorithms gave the same value, so this is a good indication. The results have been included below, with just a single value given (number of inversions) for each row in the test file. This test can be run by calling the function "test_correctness2("test_in.txt")".

Results: 252180, 250488, 243785, 247021, 250925, 256485, 249876, 253356, 255204, 247071

Extrapolation and Interpretation

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$^y - intercept * n^s lope$

Brute Force

- slope: 2.05692581355
- y-intercept: -17.5588327907

Naive Divide & Conquer:

- slope: 2.03626014268
- y-intercept: -17.3982172216

Merge & Count:

- slope: 1.10025742067
- y-intercept: -13.1947528938

Largest input item solvable in an hour:

Extrapolation using the best fit function from the previous section:

f(n) = Runtime = 1 hr = e$^{y} - intercept * n^{s}lope$

Solving for n, using the values from the above chart, yields the following numbers:

Brute Force: 122,802,000

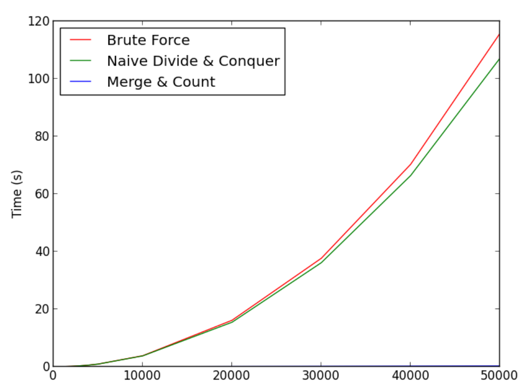
Naive Divide & Conquer:

Merge & Count:

Discrepancy between actual and asymptotic:

Empirical Analysis of Run Time

Linear Plot:



Log-log Plot: