Mitchell Van Braeckel (mvanbrae@uoguelph.ca) 1002297
Feb 11th, 2019
CIS*3490: Analysis and Design of Computer Algorithms – A2

1. (40%) A[0..n-1] is an array of *n* distinct numbers. A pair of array members (A[i], A[j]) is called an inversion if A[i] > A[j] for i < j

1.1 Design a brute force algorithm to count the number of inversions in an array, analyze the number of executions of its basic operation, and determine the efficiency class

Pseudocode:

```
SelectionCount(A[0..n-1])
      Count = 0;
      For i ← 0 to n-2 do
            For j ← i+1 to n-1 do
                  If A[i] > A[j]
                        Count++
```

Number of Executions of Basic Operation: Comparison (>)

$$\sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2}[(n-1)-(i+1)+1] = (n-1)\sum_{i=0}^{n-2} 1 - \sum_{i=0}^{n-2} i = (n-1)(n-2+1) - \sum_{i=1}^{n-2} i + 0$$
$$= \frac{2(n-1)^2}{2} - \frac{(n-2)(n-1)}{2} = \frac{(2n^2-4n+2)-(n^2-3n+2)}{2} = \frac{n^2-n}{2}$$
$$= \boldsymbol{\frac{n(n-1)}{2}}\ \boldsymbol{number\ of\ executions}$$

Therefore, its efficiency class is Θ (n^2).

1.2 Design a recursive divide-and-conquer algorithm of Θ (*n log n*) to count the number of inversions in an array, create a recurrence to analyze the number of executions of its basic operation of the best case, and determine the efficiency class. Use the Master Theorem to verify the efficiency class in your analysis result.

Pseudocode:

```
MergeSort(A[0..n-1], temp[0..n-1], left, right, *count)
      If left < right
            mid ← left + (right-left)/2
            MergeSort(A, temp, left, mid, count)     //left half
            MergeSort(A, temp, mid+1, right, count) //right half
            Merge(A, temp, left, mid, right, count) //merge back
```

```
Merge(A[0..n-1], temp[0..n-1], left, mid, right, *count)
      i ← left, j ← mid, k ← left
      while i <= mid-1 && j <= right      // n/2
            if A[i] <= A[j]
                  temp[k] ← A[i]; k ← k+1; i ← i+1
            else
                  temp[k] ← A[j]; k ← k+1; j ← j+1
      while i <= mid-1
            temp[k] ← A[i]; k ← k+1; i ← i+1
      while j <= right
            temp[k] ← A[j]; k ← k+1; j ← j+1
      for i ← left to right do     // n
            A[i] ← temp[i]
```

Number of Executions of Basic Operation During Best Case: Comparison

$n = 2^k$

$C(n) = 2\,C\left(\frac{n}{2}\right) + \frac{n}{2} \quad ; for\ n > 1$

$C(1) = 0$

Back substitution: $\quad ;\ C\left(\frac{n}{2}\right) = 2C\left(\frac{n}{4}\right) + \frac{n}{4}$

$C(n) = 2\left[2C\left(\frac{n}{4}\right) + \frac{n}{4}\right] + \frac{n}{2} = 2^2 C\left(\frac{n}{2^2}\right) + \frac{2n}{2} \quad ;\ C\left(\frac{n}{2^2}\right) = 2C\left(\frac{n}{2^3}\right) + \frac{n}{2^3}$

$C(n) = 2^2\left[2C\left(\frac{n}{2^3}\right) + \frac{n}{2^3}\right] + \frac{2n}{2} = 2^3 C\left(\frac{n}{2^3}\right) + \frac{3n}{2} \quad etc\ \ldots$

Recurrence Relation: $C\left(2^{k-i}\right) = 2^i\,C\left(\frac{n}{2^i}\right) + i\left(\frac{n}{2}\right) \quad ; for\ 1 \le i \le n$

$We\ need\ k - i = 0\ ; \quad n = 2^k, k = log_2 n$

Thus, $C(n) = 2^k C\left(2^{k-k}\right) + k(\frac{n}{2})$

$C(n) = log_2 n\,C(1) + log_2 n\left(\frac{n}{2}\right) = \mathbf{\frac{1}{2} n\ log_2 n}\ \ \boldsymbol{number\ of\ executions}$

Therefore, the basic efficiency class for best case is Ω (n log n)

<u>Checking with Master Theorem</u>

$C_{best}(n) = 2\,C_{best}\left(\frac{n}{2}\right) + \frac{n}{2}$

$f(n) = \frac{n}{2}\ \epsilon\ \Theta(n^1)$

$a = 2, b = 2, d = 1$

$Since: a = b^d = 2$

$C_{best}(n)\ \epsilon\ \Theta(n^d \log n) = \mathbf{\Theta(n^1 \log n)}$ **Thus, verified with Master Theorem**

1.3 Implement the two algorithms, and test them by using *data_1.txt*, which includes 50,000 integers. Your programs are required to display the number of inversions and execution time. Compare the differences in execution time and theoretical analysis

Brute Force = ~8000 milliseconds | Recursive DAC = ~20 milliseconds

Thus, Brute Force takes ~400 times longer to run compared to Recursive DAC. Based on efficiency classes of $\Theta(n^2)$ vs $\Theta(n \log n)$: $50000^2 = 2.5 * 10^9$ VS $50000 \log(50000) = \sim 2.3 * 10^5$ ➔ therefore, brute force method is approximately 10,000 slower theoretically as well. In conclusion, although the theoretical vs experimental values don't line up exactly due to other factors (eg. Computer/function optimization and memory operations), the latter is obviously much more efficient. Also, take note of theoretical efficiency class being log-base 10 instead of the realistic log-base 2

2. (60%) The convex hull of a set S is the smallest convex set containing S. (You can find more about the convex hull problem on pages 109-113 in the textbook). It is assumed that not all the points in S are on a straight line.

2.1 Design a brute force algorithm to solve the convex-hull problem and analyze its efficiency.

Pseudocode:

```
BruteHull(A[0..n-1], R[0..n-1])
      Count = 0;
      For i ← 0 to n-1 do
            For j ← 0 to n-1 do
                  If A[i] == A[j] continue            //skip, same point
                  For k ← 0 to n-1 do
                        If A[k] == A[i] || A[k] == A[j] continue //skip, =pt
                        If d >= 0 // A[k] NOT on LS of line from A[i] to A[j]
                              all_left =false;
                              Break
                  If all_left == true                 //add to results
                        If R does not contain A[i]
                              R[h] ← A[i]; h ← h+1
                        If R does not contain A[j]
                              R[h] ← A[j]; h ← h+1
```

Number of Executions of Basic Operation: Comparison (>=)

$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\sum_{k=0}^{n-1} 1 = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}[(n-1)+1] = \sum_{i=0}^{n-1}(n\sum_{j=0}^{n-1} 1) = \sum_{i=0}^{n-1}[n(n-1+1)] = n^2\sum_{i=0}^{n-1} 1 = n^2[n-1+1]$$
$$= \sim \boldsymbol{n^3\ number\ of\ executions}$$

Therefore, its efficiency class is Θ (n^3).

2.2 Design a recursive divide-and-conquer algorithm of Θ(n log n) to solve the convex-hull problem, create a recurrence to analyze the number of executions of the basic operation of the base case, and determine the efficiency class. Use the Master Theorem to verify the efficiency class in your analysis result.

Pseudocode:

```
QuickHull(A[0..n-1], R[0..n-1], Point p1, Point p2, side, *count)
      ind ← -1, max_dist ← 0
      For i ← 0 to n-1 do
            Temp ← distance(p1,p2,A[i])
            If which_side(p1,p2,A[i]) == side && temp > max_dist
                  ind ← i, max_dist ← temp
      if ind == -1                                  // no EP
            if R does NOT contain p1
                  R[*count] ← p1; *count ← (*count)+1
            If R does NOT contain p2
                  R[*count] ← p2; *count ← (*count)+1
      QuickHull(A,R,A[ind],p1, -1*which_side(A[ind],p1,p2),count)
      QuickHull(A,R,A[ind],p2, -1*which_side(A[ind],p2,p1),count)
```

Number of Executions of Basic Operation During Best Case: Comparison

$n = 2^k$

$C(n) = 2\,C\left(\frac{n}{2}\right) + n \quad ; for\ n > 1$

$C(0) = 0\,; C(1) = 0$

Back substitution: $; \ C\left(\frac{n}{2}\right) = 2C\left(\frac{n}{4}\right) + \frac{n}{2}$

$C(n) = 2\left[2C\left(\frac{n}{4}\right) + \frac{n}{2}\right] + \frac{n}{2} = 2^2 C\left(\frac{n}{2^2}\right) + \frac{3n}{2} \quad ; \ C\left(\frac{n}{2^2}\right) = 2C\left(\frac{n}{2^3}\right) + \frac{n}{2^2}$

$C(n) = 2^2\left[2C\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\right] + \frac{3n}{2} = 2^3 C\left(\frac{n}{2^3}\right) + \frac{5n}{2} \quad etc \ldots$

Recurrence Relation: $C\left(2^{k-i}\right) = 2^i\,C\left(\frac{n}{2^i}\right) + (2i+1)\left(\frac{n}{2}\right) \quad ; for\ 1 \le i \le n$

$We\ need\ k - i = 0 \ ; \quad n = 2^k, k = log_2 n$

Thus, $C(n) = 2^k C\left(2^{k-k}\right) + (2k+1)\left(\frac{n}{2}\right)$

$C(n) = log_2 n\,C(1) + (2log_2 n + 1)\left(\frac{n}{2}\right) = \boldsymbol{n\,log_2 n + \frac{n}{2}\ number\ of\ executions}$

Therefore, the basic efficiency class for best case is Ω (n log n)

Checking with Master Theorem

$C_{best}(n) = 2\,C_{best}\left(\frac{n}{2}\right) + n$

$f(n) = n\,\epsilon\,\Theta(n^1)$

$a = 2, b = 2, d = 1$
$Since: a = b^d = 2$
$C_{best}(n)\ \epsilon\ \Theta(n^d \log n) = \boldsymbol{\Theta(n^1 \log n)}$ **Thus, verified with Master Theorem**

2.3 Implement the two algorithms, and test them by using *data_2.txt*, which includes 30,000 points (pairs of x-y coordinates). Your programs are required to display the number of points on the convex hull and socsexecution time. Compare the differences in execution time and theoretical analysis

Brute Force = ~45000 milliseconds | Recursive DAC = ~50 milliseconds

Thus, Brute Force takes ~900 times longer to run compared to Recursive DAC. Based on efficiency classes of $\Theta(n^3)$ vs $\Theta(n \log n)$: $30000^3 = 2.7 * 10^{13}$ VS $30000 \log(30000) = \sim 1.34 * 10^5$ ➔ therefore, brute force method is approximately $2.0 * 10^8$ slower theoretically as well. In conclusion, although the theoretical vs experimental values don't line up exactly due to other factors (eg. Computer/function optimization and memory operations), the latter is obviously much more efficient. Also, take note of theoretical efficiency class being log-base 10 instead of the realistic log-base 2