CSCI-UA.0310-004/005 Basic Algorithms February 12, 2015

Solutions to Problem 1 of Homework 2 (10 points)

Name: Jason Yao *Due: Wednesday, February 11*

Let $A[1, \dots, n]$ be an array of n distinct numbers. If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an **inversion** of A.

(a) (2 points) List the five inversions of the array $\langle 2, 3, 8, 6, 1 \rangle$.

Solution: (3, 4), (1, 5),(2, 5),(3, 5), (4, 5) □

(b) (3 points) Which arrays with elements from the set $\{1, 2, \dots, n\}$ have the smallest and the largest number of inversions and why? State the expressions *exactly* in terms of n.

Solution:

Smallest number of inversions:

Array is sorted in ascending order, which by definition has no inversions. $\Theta = (n)$

Largest number of inversions:

Array is sorted in descending order, which by definition means that there are n + (n - 1) + ... + 1 inversions, and thus $\Theta = (n^2)$ □

(c) (5 points) What is the relationship between the running time of INSERTION_SORT and the number of inversions I in the input array? *Justify your answer.*

Solution: Insertion sort method relies upon the idea of swapping two elements in an array if there is an inversion. Thus, the more inversions there are in the array, the longer the running time. Conversely, fewer inversions in the array results in a shorter running time □

CSCI-UA.0310-004/005 Basic Algorithms February 12, 2015

Solutions to Problem 2 of Homework 2 (16 points)

Name: Jason Yao *Due: Wednesday, February 11*

Let $A[1 \dots n]$ be an array of pairwise different numbers. We call pair of indices $1 \leq i < j \leq n$ an *inversion* of A if $A[i] > A[j]$. The goal of this problem is to develop a divide-and-conquer based algorithm running in time $\Theta(n \log n)$ for computing the number of inversions in A.

(a) (8 points) Suppose you are given a pair of *sorted* integer arrays A and B of length $n/2$ each. Let C an n-element array consisting of the concatenation of A followed by B. Give an algorithm (in pseudocode) for counting the number of inversions in C and analyze its runtime. Make sure you also argue (in English) why your algorithm is correct.

Solution:

```
inversionCount(A[], B[], C[])
i = 0
j = A.length
counter = 0

while (i < A.length) && (j < C.length)
  if C[i] > C[j]
    ++counter
    ++j
  endif
  else // No more inversions occurring, can skip the rest
    ++i
    j = A.length // Resets for new round of comparisons
  endelse
endwhile

return counter
```

Runtime:

The algorithm's worst-case runtime is $T(n) = \Theta(n^2)$, since the while loop acts the same way as an outer for loop in a nested version of the code, in this case with resetting j.

Algorithm Correctness:

The algorithm depends upon the property of C[] in that it contains two pre-sorted arrays, meaning that it is possible to skip checking after an inversion check fails, since every other check after that would be unnecessary.

Thus, all the algorithm is doing is running a counter, and running until the first inversion check fails, then resetting, and doing it again with a different i value. □

(b) (8 points) Give an algorithm (in pseudocode) for counting the number of inversions in an n element array A that runs in time $\Theta(n \log n)$. Make sure you formally prove that your algorithm runs in time $\Theta(n \log n)$ (e.g., write the recurrence and solve it.)
(**Hint**: Combine Merge Sort with part (a).)

Solution:

```
mergedInversionCountWrapper(A[])
  mergedInversionCountRec(A, 0, A.length − 1)
end mergedInversionCounterWrapper

mergedInversionCountRec(A[], start, end)
  // Base case
  if start == end
    return
  endif

  // Recursive case
  mid = (end − start)/2 + start
  mergedInversionCountRec(A, start, mid)
  mergedInversionCountRec(A, mid + 1, end)
  mergeCount(A, start, mid, end)
end mergedInversionCountRec

mergeCount (A[], start, mid, end)
  i = start
  j = end
  counter = 0

while (i < mid) && (j < end)
  if C[i] > C[j]
    ++counter
    ++j
  endif
  else // No more inversions occurring, can skip the rest
    ++i
    j = mid // Resets for new round of comparisons
  endelse
endwhile

return counter
end mergeCount
```

□

CSCI-UA.0310-004/005 Basic Algorithms February 12, 2015

Solutions to Problem 3 of Homework 2 (15 Points)

Name: Jason Yao *Due: Wednesday, February 11*

Consider the following recurrence $T(n) = 4T(n/2) + n^2 \log n$, $T(1) = 1$.

(a) (2 points) Can the master's theorem, as stated in the book, be applied to solve this recurrence? If yes, apply it. If not, formally explain the reason why.

Solution:

$a = 4, b = 2, f(n) = n^2 \log n$

$n^{\log_2 4}\ ?\ n^2 \log n$

$n^2\ ?\ n^2 \log n$

As this case falls between cases 2 and 3 ($f(n)$ is larger than $n^{log_b a}$, but not polynomially larger), the master theorem cannot be used to solve this recurrence. $\square$

(b) (4 points) Solve the above recurrence using the recursion tree method.

Solution:

Assumptions: n is an exact power of 2

$T(n) = cn^2 \log n + 2cn^2 \log n + 4cn^2 \log n + ... + 2^n cn^2 \log n$

$T(n) = \sum_{i=1}^{n} 2^i cn^2 \log n + \Theta(n^2)$

$T(n) = O(2^n)$ $\square$

(c) (4 points) Formally verify that your answer from part (b) is correct using induction.

Solution:

$T(n) \leq c * 2^n$

Base case: n = 1

T(1) = 1 ? $\leq c * 2^{(1)}$

Okay as long as $c \geq 1$

Inductive Step:

Assume true for 1, 2, ..., n

$T(n) = 4T(n/2) + n^2 \log n \leq 4(c * 2^n) + (c * 2^n)^2 \log c * 2^n$

$n^2 ? \leq 2^n$ $\square$

(d) (5 points) Solve the above recurrence exactly using domain-range substitution.

Solution:

Domain Substitution:

$T(n) = aT(\frac{n}{b}) + f(n)$

$T(n) = 4T(n/2) + n^2 \log n$, $T(1) = 1$

Let n = 2^k

$T(b^k) = 4T(\frac{2^k}{2}) + (2^{2k})k \log 2$

$S(k) = 4S(2^{k-1}) + (2^{2k})k, S(0) = T(1) = 1 = f(1)$

Applying Range Substitution:

$R(n) = \frac{S(k)}{2^k} = \frac{4S(2^{k-1})+(2^{2k})k}{2^k} = 2^{2-k}S(2^{k-1}) + (2^k)k$

$S(k) = (2^{2k+1})k \log 2$

$k = \log n$

$T(n) = S(\log n) = \Theta(2^n)$ $\square$

CSCI-UA.0310-004/005 Basic Algorithms February 12, 2015

Solutions to Problem 4 of Homework 2 (5 points)

Name: Jason Yao *Due: Wednesday, February 11*

Solve *precisely* the recurrence $T(n) = T(\sqrt{n}) + \log n$, with $T(2) = 2$ (assume n is such that $\sqrt{\sqrt{\ldots\sqrt{n}}}$ is always an integer). (**Hint**: Substitute $n = 2^k$... Then substitute again.)

Solution:

$T(n) = T(\sqrt{n}) + \log n, T(2) = 2$
Substitute $n = 2^k (k = \log_2 n)$
$T(n) = T(2^k) = T(\sqrt{2^k}) + \log_2 2^k$
$T(2^k) = T(\sqrt{2^k}) + k$
$S(k) = S(k/2) + k, S(1) = 1$
Substitute $k = 2^m (m = \log_2 k)$
$S(k) = S(2^m) = S(\frac{2^m}{2}) + 2^m$
$S(2^m) = S(2^{m-1}) + 2^m$
$R(m) = R(m-1) + 2^m, R(0) = 1$
$R(m) = 2^{m-1} - 1$
Back-solving, $m = \log_2 k$
$R(\log_2 k) = 2^{\log_2 k+1} - 1$
$R(\log_2 k) = 2k - 1$
Back-solving, $k = \log_2 n$
$R(\log_2 \log_2 n) = 2(\log_2 n) - 1$
$T(n) = R(\log_2 \log_2 n) = 2 \log_2 n - 1$
$T(n) = 2 \log_2 n - 1$ □