

YORK UNIVERSITY – CSE 3101 – JUNE 6, 2006  
**TEST 1**

Student Number: \_\_\_\_\_

Email Address: \_\_\_\_\_

Last (Family) Name(s): \_\_\_\_\_

First (Given) Name(s): \_\_\_\_\_

- This test is worth 20% of your final mark.
- Answer each question directly on the test paper, in the space provided. Use the reverse side of the pages for rough work. If you need more space for one of your solutions, use the reverse side of a page and *indicate clearly the part of your work that should be marked*.
- **20% rule:** If you are unable to answer (part of) a question you will get 20% of the marks for the (part of the) question if you write “I don’t know” and *nothing else* for that part/question.

### The Master Theorem

If for some  $a, b, d > 0$ :

$$T(n) = aT(n/b) + \mathcal{O}(n^d)$$

then

$$T(n) = \begin{cases} \mathcal{O}(n^d) & \text{if } a < b^d \\ \mathcal{O}(n^d \log(n)) & \text{if } a = b^d \\ \mathcal{O}(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

Question 1 (out of 28)	
Question 2 (out of 32)	
Question 3 (out of 10)	
Question 4 (out of 10)	
Question 5 (out of 20)	
<b>Total (out of 100)</b>	

### Question 1 [28]

Consider the following problem, known as the “Counting Inversion Problem”. The input is an array of  $n$  *distinct* natural numbers  $A[1 \dots n]$ . *Throughout this question you can assume that  $n$  is a power of 2.* An inversion in this array is a pair  $\langle i, j \rangle$  where  $i < j$  and  $A[i] > A[j]$ . For example, in the following array

A	3	9	5	2
	1	2	3	4

there are 4 inversions:  $\langle 1, 4 \rangle$ ,  $\langle 2, 3 \rangle$ ,  $\langle 2, 4 \rangle$ , and  $\langle 3, 4 \rangle$ .

**a [2]** How many inversions are there, if  $A[1] < A[2] < \dots < A[n]$  ?

**b [2]** How many inversions are there, if  $A[1] > A[2] > \dots > A[n]$  ?

Now we want to design a divide-and-conquer algorithm for solving this problem. The idea is to divide the array into two halves, then count the number of inversions in each half, and count the number of inversions across the two halves, and return their sum.

**c [2]** If we count the number of inversions across two halves using the brute-force way, how many pairs  $\langle i, j \rangle$  do we have to check ? Write your answer as a function of the length  $n$  of  $A$ .

**d [3]** If you design a divide-and-conquer algorithm where counting the number of inversions across the two halves is done using the brute-force method, what will be the running time of the algorithm ? Give your answer using the  $\Theta$  notation. Clearly state how you apply the Master Theorem.

Counting the number of inversions across two halves can be done much faster if the two halves are already sorted in increasing order. Therefore we will also sort the input array as we compute the number of inversions. This reminds us of the Mergesort algorithm; in fact, we need just to modify the Mergesort algorithm slightly to obtain the current algorithm.

Let  $\text{Merge-and-Count}(A, p, q, r)$  be the procedure that, assuming  $A[p \dots q]$  and  $A[(q+1) \dots r]$  are already sorted in increasing order, merges  $A[p \dots r]$  into increasing order, and returns the numbers of inversions  $\langle i, j \rangle$  where  $p \leq i \leq q$  and  $q + 1 \leq j \leq r$ .

**e [4]** Using  $\text{Merge-and-Count}$ , write a recursive program called  $\text{Sort-and-Count}(A, p, r)$  that takes an input array  $A$ , sorts  $A[p \dots r]$  into increasing order, and returns the number of inversions in  $A[p \dots r]$ . (Thus, to count the number of inversions in the whole array  $A$  we call  $\text{Sort-and-Count}(A, 1, n)$ .)

**f [2]** What is the running time of  $\text{Merge-and-Count}$  ? Give your answer using the  $\Theta$  notation.

**g [10]** Give an implementation of Merge-and-Count. Your implementation must be asymptotically faster than the brute-force approach discussed before.

**h [3]** What is the running time of Sort-and-Count ? Give your answer using the  $\Theta$  notation. Clearly state how you apply the Master Theorem.

**Question 2 [32]**

**a [3]** Give the definition of  $f(n) = \mathcal{O}(g(n))$ .

**b [3]** Give the definition of  $f(n) = \omega(g(n))$ .

**c [5]** For any function  $g(n)$ , show that there is no function  $f(n)$  that is both  $o(g(n))$  and  $\omega(g(n))$ .

**d [12]** Check the boxes that apply (“IDK” stands for “I don’t know”). Note that there are four questions on each row.

$f(n)$	$g(n)$	$f(n) = \mathcal{O}(g(n))$			$f(n) = \Omega(g(n))$			$f(n) = o(g(n))$			$f(n) = \omega(g(n))$		
		True	False	IDK	True	False	IDK	True	False	IDK	True	False	IDK
$n^{100}$	$1.7^n$												
$n^{1.4}$	$n(\log n)^2$												
$3^n + 2^n$	$3^n$												

**e [5]** Show that a binary tree of height  $h$  has at most  $2^h$  leaves.

Parts **f**) and **g**) below refer to the following algorithm for Primality testing:

**Input:** A natural number  $n$  in binary notation.

1.  $m \leftarrow \lfloor \sqrt{n} \rfloor$
2. For  $i = 2 \dots m$  do
3.     If  $i|n$  then Output “No”; End If
4. End For
5. Output “Yes”

Here  $i|n$  is a symbol for “ $i$  divides  $n$ ”. Assume that computing  $\lfloor \sqrt{n} \rfloor$  and checking  $i|n$  take constant times.

**f [2]** What is the running time of the algorithm as a function of  $n$  ? (Give your answer using the  $\Theta$  notation.)

**g [2]** Let  $\ell$  be the length of the binary representation of  $n$ . What is the running time of the algorithm as a function of  $\ell$  ? (Give your answer using the  $\Theta$  notation.)

### Question 3 [10]

**a [3]** What is the “heap property” ?

Parts **b)** and **c)** below refer to the following array, which occurs at some time during the execution of the Heapsort algorithm, when the subroutine Build-Heap has completed:

A	18	17	16	15	14	13	12	11	19	20	21	22
	1	2	3	4	5	6	7	8	9	10	11	12

**b [4]** What is the current heap size ? Briefly explain.

**c [3]** What will Heapsort do next ? (Give the next step in the Heapsort algorithm.)

### Question 4 [10]

Consider the following 6 numbers in binary:

11001  
01101  
11010  
10110  
00011  
01100

We want to sort these numbers into increasing order from top to bottom. Tracing the execution of Radix sort:

**a** [5] For this part, the counting sort algorithm is to sort the numbers digit by digit, where each digit consists of just *one* bit. Write down the numbers in the order they appear after each call to Counting sort.

**b** [5] For this part, each digit consists of *two* bits. Write down the numbers in the order they appear after each call to Counting sort.

### Question 5 [20]

**a** [4] For each of the following sorting algorithms, indicate whether it uses the divide-and-conquer technique. Circle your answer:

Insertion sort:    Yes    No    I don't know

Radix sort:        Yes    No    I don't know

Quick sort:        Yes    No    I don't know

Heap sort:         Yes    No    I don't know

**b** [4] For each of the following sorting algorithms, indicate whether it is “in-place”. Circle your answer:

Insertion sort:    Yes    No    I don't know

Merge sort:        Yes    No    I don't know

Quick sort:        Yes    No    I don't know

Heap sort:         Yes    No    I don't know

For parts **c**) and **d**) below, recall the procedure Partition that is used in the Quicksort algorithm:

**Partition**( $A, p, r$ )

1.  $x \leftarrow A[r]$       % The “pivot” element
2.  $i \leftarrow p - 1$
3. For  $j = p \dots (r - 1)$  do
4.     If  $A[j] \leq x$  then do
5.          $i \leftarrow i + 1$
6.         exchange  $A[i]$  and  $A[j]$
7.     End If
8. End For
9. exchange  $A[i + 1]$  and  $A[r]$
10. return  $(i + 1)$

**c** [2] What does Partition( $A, p, r$ ) return when all elements in the array  $A[p \dots r]$  have the same value ?

**d** [5] What is the running time of Quicksort if all elements in the input array  $A[1 \dots n]$  are the same ?

**e [5]** Modify Partition so that after partitioning, all elements that have the same value as the pivot element are in a contiguous block  $A[q \dots s]$ , and that Partition outputs the middle index of this block (i.e.,  $\lceil \frac{q+s}{2} \rceil$ ).