CS251 Homework 1

Handed out: Feb 20, 2017
Due date: Feb 27, 2017 at 11:59pm (This is a **FIRM** deadline, solutions will be released immediately after the deadline)

Question	Topic	Point Value	Score
1	True / False	5	
2	Match the Columns	7	
3	Short Answers	22	
4	Programming Questions	22	
5	Symbol Tables	4	
Total		60	

1. True/False [**5 points**]

F 1. Amortized analysis is used to determine the worst case running time of an algorithm.

F 2. An algorithm using $5n^3 + 12n\log n$ operations is a $\Theta(n\log n)$algorithm.

F 3. An array is partially sorted if the number of inversions is linearithmic.

T 4. Shellsort is an unstable sorting algorithm.

T 5. Some inputs cause Quicksort to use a quadratic number of compares.

2. Match the columns [**7 points**]

A. Mergesort	1. Works well with duplicates
B. Quicksort	2. Optimal time and space
C. Shellsort	3. Works well with order
D. Insertion sort	4. Not analyzed
E. Selection sort	5. Stable and fast
F. 3-way quicksort	6. Optimal data movement
G. Heapsort	7. Fast general-purpose sort

3. Short Answers [**22 points**]

(a) Suppose that the running time T(n) of an algorithm on an input of size n satisfies $T(n) = T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + cn$ for all n > 2, where c is a positive constant. Prove that $T(n) \sim cn\log_2 n$. [**4 points**]

$$T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) \le T(n) \; and \; T\left(\left\lceil \frac{n}{2} \right\rceil\right) \ge T(n)$$
$$T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + 2 \le T(n) \; and \; T\left(\left\lceil \frac{n}{4} \right\rceil\right) + 2 \ge T(n)$$
$$T\left(\left\lfloor \frac{n}{8} \right\rfloor\right) + 2 + 2 \le T(n) \; and \; T\left(\left\lceil \frac{n}{8} \right\rceil\right) + 2 + 2 \ge T(n)$$
$$T\left(\left\lfloor \frac{n}{n} \right\rfloor\right) + 2 + 2 \ldots + 2 \; \le T(n) \; and \; T\left(\left\lceil \frac{n}{n} \right\rceil\right) + 2 + 2 \ldots + 2 \ge T(n)$$
$$2 + \log(n) \; and \; 2 + \log_2(n)$$

$$T(n) \sim cn\log_2 n$$

(b) Rank the following functions in increasing order of their asymptotic complexity class. If some are in the same class indicate so. [**4 points**]

• $n \log n$	3.
• $n^2/201$	4. Same as $n(n-1)+3n$
• n	2.
• $\log^7 n$	1. Best Complexity
• $2^{n/2}$	5. Worst Complexity
• $n(n-1)+3n$	4. Same as $n^2/201$

(c) Consider the following code fragment for an array of integers:

```
int count = 0;
int N = a.length;
Arrays.sort(a);
for (int i = 0; i < N; i++)
  for (int j = i+1; j < N; j++)
    for (int k = j+1; k < N; k++)
      if (a[i] + a[j] + a[k] == 0)
        count++;
```

Give a formula in tilde notation that expresses its running time as a function of N. If you observe that it takes 500 seconds to run the code for N=200, predict what the running time will be for N=10000. [**5 points**]

$$Nlog(N) + (N)\cdot(N-1)\cdot(N-2)$$

$$Nlog(N) + \binom{1}{3}N = \sim\frac{N^3}{6}$$

$$500s = c\cdot\frac{200^2}{6}; c = \frac{3}{40}s$$

Using n=10,000 with a constant of $\frac{3}{40}$s would take about 1,250,000 seconds or 14.47 days.

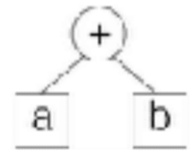
(d) In Project 2 you were asked to use `Arrays.sort(Object o)` because this sort is stable. What sorting algorithm seen in class is used in this case? What sorting algorithm would you use if instead of dealing with `Point` objects you were handling `float` values? Justify your answer. [**4 points**]

ANSWER

Mergesort is used for Arrays.sort(Object) since its stability is required when dealing with objects. The only other stable sort, Insertion sort, should only be used for a small N value whereas Mergesort guarantees Nlog(n) performance.

If handling float values, or some other primitive type where stability is not as much of a concern, you would use quick sort which is the fastest sort in practice.

(e) Convert the following (*Infix)* expressions to *Postfix* and *Prefix* expressions (To answer this question you may find helpful to think of an expression "a + b" as the tree below.) [**5 points**]



(i) (a + b) * (c / d)

Postfix (RPN): (a b +) (c d /) *

Prefix: * (+ a b) (/ c d)

(ii) a * (b / c) - d * e

Postfix: (b c /) a * (d e *) -

Prefix: - * (/ b c) a (* d e)

(iii) a + (b * c) / d - e

Postfix: a (b c *) d / + e -

Prefix: - / + a (* b c) d e

(iv) a * b + c * (d / e)

Postfix: a b * c + (d e /) *

Prefix: * ab + c * (/ d e)

(v) a * (b / c) + d / e

Postfix: a (b c /) * d e / +

Prefix: + / a (/ b c) * de

4. Programming Questions [**22 points**]

(a) Give the pseudocode to convert a fully parenthesized expression (*i.e.*, an INFIX expression) to a POSTFIX expression and then evaluate the POSTFIX expression. [**5 points**]

```
function toPostfix(char[] s)
   Stack stack;
   StringBuilder postfix;
   for (i → s.length where c : s[i])
      if (c is operator)
         postfix += c;
      while (importance(c) <=
         importance(stack.top))
         postfix += pop(stack);
      push(c);
   while (stack not empty)
      postfix += pop(stack);
   return postfix;
```

```
function evalPostfix(string s)
   Stack stack;
   for (i → s.length where c : s[i])
      if (c is operator)
         var obj1 = pop(stack);
         var obj2 = pop(stack);
         var obj3 = operation(obj1,
                              obj2,
                              c);
         push(obj3);
      else
         push(c)
   print pop(stack);
```

(b) Given two sets A and B represented as sorted sequences, give Java code or pseudocode of an efficient algorithm for computing A ⊕ B, which is the set of elements that are in A or B, but not in both. Explain why your method is correct. [**5 points**]

ANSWER TO (b)

```
function unionMinusIntersection (Obj[] a, Obj[] b)
   Dictionary<int, Boolean> stored = new Dictionary(a.length
                                                   + b. length);
   removeDuplicates(a);
   removeDuplicates(b);
   for (i → a.length where x : a[i])
      if (stored.contains(x))
         stored.add(x, true)
      else
         stored.add(x, false)
   for (i → b.length where x : b[i])
      if (stored.contains(x))
         stored.add(x, true)
      else
         stored.add(x, false)
   for (KeyValuePair<int, Boolean> pair : stored)
      if (pair.value == false)
         print pair.key
```

JUSTIFICATION

The above algorithm keeps track of whether a value is stored in A or B by simply updating a Boolean value indicating whether or not It's a duplicate value. If it's a duplicate value, specifically both in A and B, we ignore reading it entirely. Thus producing an output free of duplicates of any kind.

(c) Let A be an unsorted array of integers $a_0, a_1, a_2, \ldots, a_{n-1}$. An inversion in A is a pair of indices (i, j) with i < j and $a_i > a_j$. Modify the merge sort algorithm so as to count the total number of inversions in A in time $\mathcal{O}(n \log n)$. [**5 points**]

```
static int swapCount = 0;

Function merge(int low, int middle,
int high)
   Array copy = array.copy from low to
high

   int i = low, j = middle + 1
   int k = high

   while (i <= middle && j <= high)
      if (copy[i] < copy[j])
         array[k] = copy[i]
         i++
      else
         array[k] = copy[j]
         add (j-k) to SwapCount
         j++
```

```
Function sort(int low, int high)
   if (low < high)
      int middle = low + (high - low) / 2
      sort(low, middle)
      sort(middle+1, high)
      merge(0, middle, high)

   return swapCount;
```

(d) Let A [1 . . . n] , B[1 . . .n] be two arrays, each containing *n* numbers in sorted order. Devise an $\mathcal{O}(\log n)$ algorithm that computes the *k*-th largest number of the 2*n* numbers in the union of the two arrays. Do not just give pseudocode — explain your algorithm and analyze its running time.

For full credit propose a solution using constant space. [**7 points**]

```
var maxA = last element of A
var maxB = last element of B

if maxA == maxB return maxA

var midA = middle element of A or -Infinity if index < 0
var midB = middle element of B or -Infinity if index < 0

if midA >= midB
       //check the upper quarter for a greater value
       var midmidA = middle element of A in upper quarter
       var midmidB = middle element of B in upper quuarer
       if midmidA >= midmidB
              //check four quarter
              search index to min(length - midmidA, length - midmidB) return highest
equal
       else
              //check third quarter
              search index to min(midmidA - midA, midmidB - midB) return highest equal
else if (idA < midB
       //check the lower quarter for a greater value
       var midmidA = middle element of A in upper quarter
       var midmidB = middle element of B in upper quuarer
       if midmidA >= midmidB
              //search second quarter
              search index to min(midA - midmidA, midB - midmidB) return highest equal
       else
              //search first quarter
              search index to min(midmidA, midmidB) return highest equal

DOUBLE CLICK
```

5. Symbol Tables [**4 points**]

Draw the Red-Black LL BST obtained by inserting following keys in the given order:
H O M E W O R K S.

Assuming that duplicate keys are not allowed