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**]

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

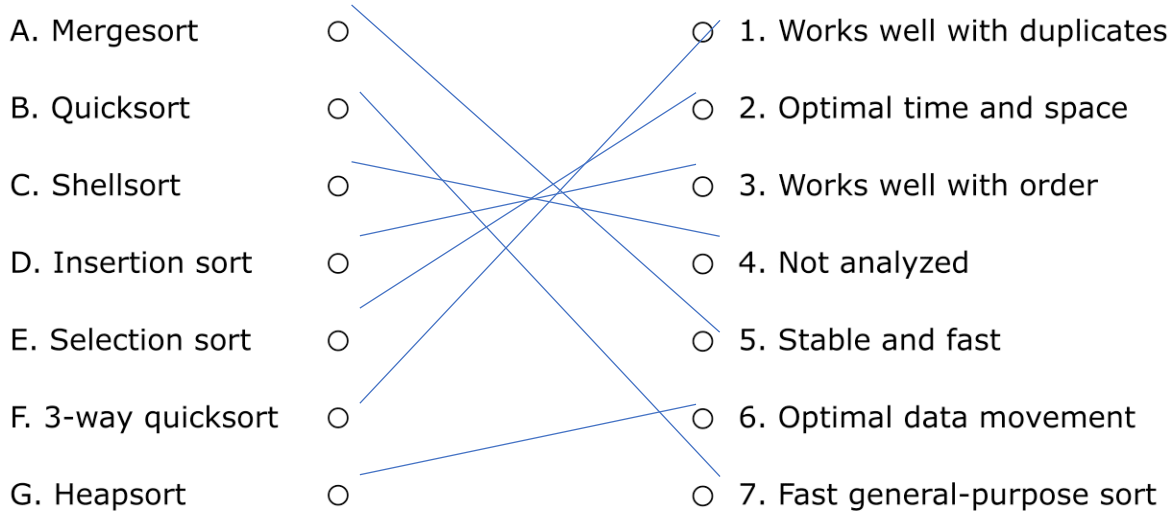
X 2. An algorithm using $5n^3 + 12n \log n$ operations is a $\Theta(n \log n)$ algorithm.

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

X 4. Shellsort is an unstable sorting algorithm.

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

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



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

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

ANSWER: Consider the following given recurrence relation: T(n) = T([n/2]upper) + T([n/2]lower) + cn
In the substitution method a solution is guessed every time and checked whether its working or not.
Now solve the recurrence relation using substitution method, by making a guess that T(n) = O(cnlog$_2$n).
Since substitution method is using, it is required to prove that T(n) <= cnlog$_2$n for C > 0.
Thus, T([n/2]upper) <= c([n/2]upper)log$_2$([n/2]upper), same for the T([n/2]lower).
Then, T(n) <= c([n/2]upper)log$_2$([n/2]upper)) + c([n/2]lower)log$_2$([n/2]lower)) + cn
=(cn/2c)log$_2$(n/2) + (cn/2c)log$_2$(n/2) + cn
=2(cn/2c)log$_2$(n/2) + cn = nlog$_2$(n/2) + cn which is ~ to cnlog$_2$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**]

- $n \log n$
- $n^2/201$
- n
- $\log^7 n$
- $2^{n/2}$
- $n(n-1)+3n$

ANSWER:
Log^7n < n < nlogn < n^2/201 < n(n-1) +3n < 2^n/2

(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++;
```

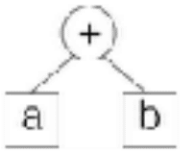
Give a formula in tilde notation that expresses its running time as a function of N. If you observe that it takes 500 seconds to run the code for N=200, predict what the running time will be for N=10000. [**5 points**]

ANSWER T(N) = N^3. If 500 seconds to run N = 200, if N = 10000, 6.25 * 10^-5 * (N^3) = 6.25 * 10^-5 * 10000^3 = 62500000 seconds.

(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: Array.sort uses MergeSort. For sorting float values, I would prefer Radix sort, because it works in O(n) and use its property of IEEE floats being sorted when their bit pattern is interpreted as ints, in another word, as sign-magnitude integers rather than two's complement integers. Another sort I would use is a combination of Heapsort and quicksort.

(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)
ANSWER: Postfix: a b + c d / *, Prefix: * + a b / c d

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

(iii) a + (b * c) / d - e
ANSWER: Postfix:a b c * d / + e – Prefix - + a / * d b c e

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

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

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**]

ANSWER:

```
For ( each char c in the in fix expression) {
 Switch(c) {
      Case operand: postfix exp = postfixexp + c; break;
      Case '(': astack.push(c); break;
      Case ' )' : while( top of stack is not '(') { postfixexp =
postfixexp + (top of aStack); aStack.pop(); } astack.pop; break;
      Case operator: while (!astack.isempty() && top of stack is not
'(' && precedence(c) <= precedence(top of astack) { postfixecp =
postfixexp + (top of stack); astack.pop();} astack,push©; break;
}}}
```

(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: My method is build on mathematical methord A ⊕ B = (A – B) ∪ (B – A) or = (A ∪ B) – (A ∩ B). The pseudocode is as follow:
For (loop through A. length as i increases) { for(loop through B. length as j increases) { Array "arraySame" will store all the elements that are same from A[i] with B[j]} NEXT PAGE

For (loop through A. length as i increases) { for(loop through B. length as j increases) { Array "arrayDif" will store all the elements that are not the same from A[i] with B[j] }
Then Loop through arraySame and ArrayDif
If element arraysame and ArrayDif has a element that are only exist in either of them
Then store in arrayResult
Return ArrayResult;

(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**]

ANSWER
```
public static int mergeSort(int[] a, int start, int end, int[] aux) {
    if (start >= end) {
       return 0;
    }
    int invCount = 0;
    int mid = start + (end - start) / 2;
    int invCountLeft = mergeSort(a, start, mid, aux); // divide and conquer
    int invCountRight = mergeSort(a, mid + 1, end, aux); // divide and conquer
    invCount += (invCountLeft + invCountRight);
    for (int i = start; i <= end; i++) {
       aux[i] = a[i];
    }
    int left = start;
    int right = mid + 1;
    int index = start;
    while (left <= mid && right <= end) {
       if (aux[left] < aux[right]) {
          a[index++] = aux[left++]; } else { a[index++] = aux[right++]; invCount += mid - left + 1; }}
         while (left <= mid) { a[index++] = aux[left++];} return invCount;
```

(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**]

ANSWER

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.

ANSWER