CSCI 6632/3326 — Algorithms — 13 Fall
HW 1, Due: 09/25/13 Wednesday

- Please read the policy on plagiarism and on homework guidelines(in the course outline handed out on the first day) and remember that I enforce that policy. **Email submissions will not be accepted.**
- The extra credit problems are to be done on separate sheets of paper and handed in during the finals week.
- Whenever you presenting an algorithm, remember that you should first give a short (2-3 lines) description of the main idea behind the algorithm before you get into the details of the pseudocode.
- If you are going to be using an algorithm discussed in class or the textbook (eg: Insertion Sort) you don't have to write the pseudocode for that algorithm; you can just refer to it by name.
- Quiz 1 (open book, open notes) will be on 10/02/13 Wednesday and will cover those topics from the following list which we cover in Weeks 1 and 2.

  Preliminaries: Review of elementary math concepts: logarithmic and exponential functions, Σ, Π, Algorithms: how to specify, correctness, efficiency, worst case, average case and best case analysis, order of growth, big-oh $O()$ notation – Chapters 1 and 3. Sorting: Insertion Sort – Chapter 2. $\Omega()$ and $\Theta()$ notation – 3.1. Merge Sort – 2.3. Counting Sort – 8.2. Review of elementary math concepts: permutations and combinations, proof by induction, proof by contradiction.

1. (20 points)
   (a) Suppose a sorting algorithm takes 3 seconds to sort 100 items. How much time in seconds will the same algorithm take to sort 1000 items if the number of operations the algorithm performs is exactly n^2.
   (b) Suppose algorithm A takes $50n\log_2 n$ time and algorithm B takes $25n^2$ time. What is the smallest value of n (assuming $n > 2$) for which A will be faster than B? It may be difficult to do an exact calculation here; if so, try to give an approximate answer.
   (c) For each of the following, indicate whether the statement is true or false (no explanations necessary):
      i. n^3 is $O(2^n)$.
      ii. $5n^3$ is $O(\log_2 n)$.
      iii. $3n^5$ is $O(n^2)$.
      iv. $82n^2 + 6n$ is $O(n^3)$.
      v. $3n^2 + 5n + 3$ is $O(n^2)$.
   (d) If $p(n)$ and $q(n)$ are two functions such that $p(n)$ is $O(q(n))$, does this always mean that for every value of n, $p(n) \leq q(n)$? If you think the answer is yes, give a justification. If you think the answer is no, give a counterexample i.e. find two functions $p(n)$ and $q(n)$ and a number y such that $p(n)$ is $O(q(n))$ and $p(y) > q(y)$.

2. (10 points)
   (a) Trace insertion sort on the list $4, 7, 2, 6, 9, 3$. i.e. show what the list looks like after each iteration of the outer loop. Also, calculate the exact number of total data comparisons made. Note that a data comparison is when two different pieces of data are compared, and not when loop variables etc are compared.
   (b) The same way as is done in the textbook, trace merge sort on the list $8, 9, 5, 6, 7, 2, 4, 1$. Also, calculate the exact number of total data comparisons made.

3. (10 points) Consider the following sorting algorithm (the UNH sort) to sort an array $A[1..n]$.

```
for i = n-1 downto 1 do
  for j = 1 to i do
     if A[j] > A[j+1] then
        swap(A[j],A[j+1])
```

   (a) Trace this sorting algorithm on the list $8, 4, 7, 2, 9, 3$. i.e. show what the list looks like after each iteration of the outer loop. Also, calculate the exact number of total data comparisons made.
   (b) As we did with insertion sort in class, do a worst case analysis (use big O notation) of the running time of this algorithm. You don't have to give any explanations here - just state your answer.
   (c) As we did with insertion sort in class, do a best case analysis (use big O notation) of the running time of this algorithm. You don't have to give any explanations here - just state your answer.

4. (20 points)

   Let A and B be two *sorted* arrays, each with n elements. You have to write an *efficient* algorithm which finds out if A and B have any elements in common.

   For example if $A = [4, 7, 12, 15]$, B = [2,4,6,23] the answer should be yes (since 4 is an element in common), if $A = [4, 7, 9]$, B = [2,3, 28] the answer should be no. In terms of efficiency, the faster your algorithm is, the better.

   - You will get full credit if your algorithm is correct and works in time $O(n)$.
   - You will get substantial partial credit if your algorithm is correct and works in time smaller than $O(n^2)$ but bigger than $O(n)$.
   - You will get some partial credit if your algorithm is correct and works in time $O(n^2)$.

   (a) Give a clear description of your algorithm for the above problem.

   (b) Trace how your algorithm works on the above two examples.

   (c) Do a worst case time analysis (use big O notation) of your algorithm.

5. (20 points) Let $A[1]$, $A[2]$, …, $A[n]$, be n integers (possibly with repetitions) in the range $1 \dots k$ i.e. all the entries of the array A are between 1 and k. You want to repeatedly answer range queries of the form $[a, b]$ where $1 \leq a \leq b \leq k$; each range query asks how many $A[i]$'s are in the range $a \dots b$, i.e., how many numbers from A lie between a and b. For example, if $k = 10$, and the array A has seven elements $5, 9, 3, 5, 10, 6, 1, 7$ the range query $[2, 7]$ asks how many numbers are there in the array A between 2 and 7, so the answer should be 5 (because the numbers 3,5,5,6,7 from A lie in the range $[2, 7]$). You want to do this efficiently. In order to do this, you first do some preprocessing, which is a one-time operation i.e. you are willing to pay the one-time relatively expensive cost of doing preprocessing before starting to answer the range queries, in order to save time with the many range queries which will follow. Your algorithm should work within the following time bounds:

   - The one time preprocessing step should take $O(n + k)$ time.
   - Each range query should be answered in $O(1)$ (i.e. constant) time i.e. the amount of time to answer the range query $[a, b]$ will be a small number which will in no way depend on n,k,A,a,b or the number of elements which actually lie in the range $[a, b]$.

   (a) Give a clear description of your algorithm for the above problem. This description should include both how the preprocessing step is carried out, and then how the range queries are answered.

   (b) Trace your algorithm on the above example i.e. for the array A as above, show what will happen in the preprocessing step. Then show how the algorithm will answer the range query $[2, 7]$. Also show how the algorithm will answer the range query $[1, 9]$.

   (c) Do a worst case time analysis (use big O notation) of your algorithm i.e. answer the two questions: how much time does the preprocessing step take and how much time does answering the range query take.

   Hint: Think about counting sort discussed in class and in the textbook. You will need to keep the auxillary array $C[1..k]$ we used for counting sort where $C[j]$ represents the number of elements from A which are less than or equal to j. In the above example, $C[2]$ will be 1 since there is one element (i.e. 1) in A which is ≤ 2; $C[7]$ will be 6 since there are 6 elements (i.e. 1,3,5,5,6,7) which are ≤ 7. In class we saw how to fill C in time $O(n + k)$ (this is the preprocessing step), and once the array C is available, you should show how to use it to answer range queries in $O(1)$ time.

6. (20 points)

   this problem is for CSCI 6632 students only, not for CSCI 3326 students.

   In this part you have to come up with an algorithm to answer a more complicated range query, namely, given an array A as above, a range query $[a, b]$ now requires you to actually list all of the $A[i]$'s in the range $a \dots b$. Using the same array A as above, the range query $[2, 7]$ asks which numbers from A are there in the array A are between 2 and 7, so the answer should be 3,5,5,6,7 (it doesn't matter in what order the numbers are outputted).

   Your algorithm should work within the following time bounds:

   - The one time preprocessing step should take $O(n + k)$ time (the preprocessing here will be different from that in the first part).

- Given a range query $[a, b]$, the time taken to answer this query should not depend on k or on n or on the quantity $b - a$; instead, the query should be answered in time $O(t)$ where t is the number of elements in the output (the number of $A[i]$'s in the range $[a, b]$). For example, if we consider two range queries, $[a_1, b_1]$ and $[a_2, b_2]$, $[a_1, b_1]$ leads to an output of 500 elements, and $[a_2, b_2]$ leads to an output of 100 elements, then answering $[a_1, b_1]$ should take approximately 5 times as much time as answering $[a_2, b_2]$. Or consider another example, where $k = 10^6$, $n = 10^5$, $a = 2000$, $b = 3000$, and there are only three elements in the range $[2000, 3000]$. If you look at every entry in C between 2000 and 3000 i.e. you look at $C[2000], C[2001], C[2002], \ldots, C[3000]$, you are looking at $b - a = 1001$ entries which is too expensive since $t = 3$ is much smaller than $b - 1$ i.e. there are only three elements in the range $[2000, 3000]$.

(a) Give a clear description of your algorithm for the above problem. This description should include both how the preprocessing step is carried out, and then how the range queries are answered.

(b) Trace your algorithm on the above example i.e. for the array A as above, show what will happen in the preprocessing step. Then show how the algorithm will answer the range query $[2, 7]$. Also show how the algorithm will answer the range query $[1, 9]$.

(c) Do a worst case time analysis (use big O notation) of your algorithm i.e. answer the two questions: how much time does the preprocessing step take and how much time does answering the range query take.

Hint: You will need to keep one or more auxillary arrays; what these are going to be, you need to figure out. You should show how to fill these auxillary array(s) in time $O(n + k)$ (this is the preprocessing step), and once these auxillary array(s) are available, you have to figure out how they can be used to quickly answer the range queries.

7. (20 points)

this problem is for CSCI 3326 students only, not for CSCI 6632 students.

You have a choice of 2 problems. You have to do one of these i.e. please do one or the other, but do not do both:

Choice 1: Do Problem 6 i.e. the 2nd range query problem which is listed as for CSCI 6632 students only.

Choice 2: Do the following problem:

Let $T[1..n]$ be an array of n integers, where n is odd. An integer is a *commonly occuring element* in T if it is equal to at least half the elements in T.

For example, if $T = 5, 6, 6, 3, 6$, then the commonly occuring element of T is 6.
For example, if $T = 3, 7, 5, 4, 3$, then T has no commonly occuring element.

You have to write an *efficient* algorithm which finds out if T has a commonly occuring element, and if so to find it. In terms of efficiency, the faster your algorithm is, the better.

- You will get full credit if your algorithm is correct and works in time $O(n \log n)$.
- You will get some partial credit if your algorithm is correct and works in time more than $O(n \log n)$

(a) Give a clear description of the basic idea behind your algorithm.

(b) Give the pseudocode.

(c) Trace the execution of your algorithms on the two examples given above.

(d) Do a worst case time analysis (use big O notation) of your algorithm.

Hint: Think about what we know how to do in $O(n \log n)$ time.

Extra Credit Problem 1: Implement (i.e. program) insertion sort, merge sort and UNH sort, run your programs on different random inputs (i.e. the inputs are generated using a random number generator) with 10,000 numbers, and compare the three algorithms on how much time they take. Hand in your code, some sample runs and the timing analysis; the timing analysis (how much time each of the sorting programs actually takes) should be presented in an easy to understand way.

Extra Credit Problem 2: Implement (program) your algorithm for the problem of finding if two sorted arrays have an element in common, and show how it works on some examples.

Extra Credit Problem 3: Implement (program) your algorithm for the first range query problem (problem 5), and show how it works on some examples.