Due: April 9th at the start of class

Homework #3

CMSC351H - Spring 2019

PRINT Name: ________________________________

- Grades depend on neatness and clarity.
- Write your answers with enough detail about your approach and concepts used, so that the grader will be able to understand it easily. You should ALWAYS prove the correctness of your algorithms either directly or by referring to a proof in the book.
- Write your answers in the spaces provided. If needed, attach other pages.
- The grades would be out of 16. Four problems would be selected and everyone’s grade would be based only on those problems. You will also get 4 bonus points for trying to solve all problems.

1. [Prob 6.1,Pg 175] Design a good strategy for the following well-known game: One player thinks of a number in the range 1 to $n$. The other player attempts to find the number by asking questions of the form “is the number less than (greater than) $x$?” The object is to asking as few questions as possible. (Assume than nobody cheats).

We can find the answer using the binary search algorithm. In each step, assuming that the answer is in the range of $[a,b)$ (which means that $a \leq ans < b$), we ask “is the number less than $(a+b)/2$?” If the answer is “yes”, we continue with the range $[a,(a+b)/2)$; if the answer is “no”, we continue with the range $[(a+b)/2,b)$. Note that the game starts with $[1,n+1)$. Therefore we ask at most $\lceil \lg n \rceil$ questions to find the answer.
2. Consider the sequence of numbers below as the input:

\[ 6 \ 2 \ 8 \ 5 \ 10 \ 9 \ 12 \ 1 \ 15 \ 5 \]

a) Sort the sequence using quicksort algorithm (Fig 6.11). Rewrite the sequence after each swap operation. What is the number of comparisons between elements of the input array used by quicksort?

48 comparisons.

\[ 6 \ 2 \ 8 \ 5 \ 10 \ 9 \ 12 \ 1 \ 15 \ 5 \]

Pivot=6; \[ 6 \ 2 \ 5 \ 5 \ 10 \ 9 \ 12 \ 1 \ 15 \ 8 \]

Pivot=6; \[ 6 \ 2 \ 5 \ 5 \ 1 \ 9 \ 12 \ 10 \ 15 \ 8 \]

Pivot=6; \[ 2 \ 5 \ 5 \ 6 \ 9 \ 12 \ 10 \ 15 \ 8 \]

Pivot=1; \[ 1 \ 2 \ 5 \ 5 \ 6 \ 9 \ 12 \ 10 \ 15 \ 8 \]

Pivot=2; \[ 1 \ 2 \ 5 \ 5 \ 6 \ 9 \ 12 \ 10 \ 15 \ 8 \]

Pivot=5; \[ 1 \ 2 \ 5 \ 5 \ 6 \ 9 \ 12 \ 10 \ 15 \ 8 \]

Pivot=9; \[ 1 \ 2 \ 5 \ 5 \ 6 \ 9 \ 12 \ 10 \ 15 \ 12 \]

Pivot=9; \[ 1 \ 2 \ 5 \ 5 \ 6 \ 8 \ 9 \ 10 \ 15 \ 12 \]

Pivot=10; \[ 1 \ 2 \ 5 \ 5 \ 6 \ 8 \ 9 \ 10 \ 15 \ 12 \]

Pivot=15; \[ 1 \ 2 \ 5 \ 5 \ 6 \ 8 \ 9 \ 10 \ 12 \ 15 \]

b) Count the number of comparisons between elements of the input array if we use mergesort (Fig 6.7).

About 22 comparisons.

c) Write the pseudo-code for insertion sort and selection sort. What is the number of comparisons used by each of these algorithms?

You can check Wikipedia for the pseudo codes. Based on the implementation, insertion sort needs something between 20 to 35 comparisons. Selection sort needs 45 comparisons in the worst case.
3. [Prob 6.4, Pg 176] Construct an example for which interpolation search will use $\Omega(n)$ comparisons for searching in a table of size $n$.

Define the input table as $x[i] = 2^{i-1}$ and the search key $z = x[n - 1] + 1 = 2^{2n-2}$. In this case the best guess in interpolation search is always $Left+1$ since $\frac{(x[Left])}{x[n] - x[Left]} \approx 2^{-2n-2} < 1$ for all $Left \in [1, n - 1]$. This means that at each iteration only one element is removed, and $\Omega(n)$ comparisons are made.
4. You are given an array of \( n \) elements, and you notice that some of them are duplicates, that is, they appear more than once in the array. Show how to remove all duplicates from the array in time \( O(n \log n) \).

Sort the elements of the array using mergesort in \( O(n \log n) \) time, and then remove elements by traversing the sorted array.
5. [Prob 6.35, Pg. 179] The *weighted selection problem* is the following. The input is a sequence of distinct numbers $x_1, x_2, \ldots, x_n$ such that each number $x_i$ has a positive weight $w(x_i)$ associated with it. Let $W$ be the sum of all weights. The problem is to find, given a value $X, 0 \leq X \leq W$, the number $x_j$ such that $\sum_{i \leq j} w(x_i) < X$, and $w(x_j) + \sum_{j < i} w(x_i) \geq X$. (Notice there is a typo in the book, it is corrected here.)

You can use the *Partition* function as in the unweighted selection problem. However, given the *Middle* position returned you should compare the sum of weights from *Left* to *Middle* and choose which partition to keep looking in. This has the same running time as the unweighted selection problem.
6. [Prob 6.23, Pg 177] Given two sets $S_1$ and $S_2$, and a real number $x$, find whether there exists an element from $S_1$ and an element from $S_2$ whose sum is exactly $x$. The algorithm should run in time $O(n \log n)$, where $n$ is the total number of elements in both sets.

We are looking for a pair $(a, b)$ such that $a \in S_1$, $b \in S_2$ and $a + b = x$. Thus if we consider an element $a$ in $S_1$, we should check if there exist an element with value $x - a$ in $S_2$. This means that we should be able to search quickly in $S_2$. Therefore, first we sort the elements of $S_2$ in time $O(n \log n)$ and then for every value $a \in S$, we check the existence of $x - a$ in $S_2$ using binary search in time $O(\log n)$. 
7. [Prob 6.24, Pg 177] Design an algorithm to determine whether two sets are disjoint. State the complexity of your algorithm in terms of the sizes $m$ and $n$ of the given sets. Make sure to consider the case where $m$ is substantially smaller than $n$.

Sort the first array (suppose of size $m$) then search the elements of the second array on the sorted first array. The running time is $O(m \log m + n \log m)$, when $m$ is much smaller than $n$ this is $\approx O(n \log m)$. This indicates that it is better to sort the smaller array.
8. Show how to sort \( n \) integers in the range 0 to \( n^2 - 1 \) in \( O(n) \) time. (Hint: Consider the numbers in base \( n \), i.e., each number \( x \) in the range \([0 \ldots (n^2 - 1)]\) can be considered in the form \((ab)_n\) where \( a = x \div n \) and \( b = x \mod n \).

If we have \( n \) numbers in base \( b \), each with at most \( d \) digits, we can sort all of them in time \( O(d(n + b)) \) by using radix sort. Using the hint, we can write all the numbers in the input in base \( n \) with at most 2 digits. Thus the running time of radix sort would be \( O(2(n + n)) = O(n) \).