CSci 4041, Midterm 1 exam

Name:

Student ID number:

Instructions: The time limit is 75 minutes. Please write your answers in the space below. If you need more space, write on the back of the paper. The exam is open book and notes. You may use electronic devices to ONLY look at either an e-book version or electronic notes. You may not use the internet, program/run code or any other outside resources. (If you are typing on your keyboard/input device for anything other than ctrl-F to find words in the e-book or notes, this is probably not acceptable.) **For all questions you must show work**.

Problem 1. (20 points)
Use radix-sort to sort (with first word in dictionary on the left) these words:
“abcd”, “aacb”, “baba”, “dada”, “bada”, “dddd”, “cacc”

(Order is important in all of these. If more work is shown but there is mistakes, less points taken off)

Original	Sort right most	right-middle	left-middle	left most
“abcd”	“baba”	“baba”	“baba”	“aacb”
“aacb”	“dada”	“aacb”	“aacb”	“abcd”
“baba”	“bada”	“cacc”	“cacc”	“baba”
“dada”	“aacb”	“abcd”	“dada”	“bada”
“bada”	“cacc”	“dada”	“bada”	“cacc”
“dddd”	“abcd”	“bada”	“abcd”	“dada”
“cacc”	“dddd”	“dddd”	“dddd”	“dddd”

Final order:
“aacb”, “abcd”, “baba”, “bada”, “cacc”, “dada”, “dddd”

Problem 2. (20 points)
Find the median of this array in O(n) time **without sorting the whole array**:
A = [5, 2, 7, 9, 4, 5, 7, 2, 8]

I will select the last element as pivot, but if you use the randomized-selection, you can pick any pivot.
9 elements, so middle is the 5th = Select(5)

Pivot = 8, will swap 9 with everything to rigth:
1 2 3 4 5 6 7 8
[5,2,7,4,5,7,2 | 8 | 9], 8 is in index 8 (starting at 1), as 5 < 8 we will recursion on the left partition

Pivot = 2, Array = [5,2,7,4,5,7,2] (start index 1). Only first 2 will swap, giving:
[2,5,7,4,5,7,2], then putting pivot in place:
[2 |2| 7,4,5,7,5], we found 2 in index 2. as 5 > 2 we will recursion on the right partition

Pivot = 5, Array = [7,4,5,7,5] (start index 3). 4 and 5 will swap, giving:
[4,5,7,7,5], moving pivot....
[4,5 | 5 | 7,7], we found 5 at index 5. This is the index we wanted, so the median is 5

If you use randomize select to "happen" to pick 5, this will result in:
(Pick 5 at index 1):
A = [5, 2, 7, 9, 4, 5, 7, 2, 8], swap pivot with last...
[8, 2, 7, 9, 4, 5, 7, 2, 5] then will swap, 2, 4, 5, and 2.
[2, 4, 5, 2, 8, 7, 7, 9, 5] then move 5 into place...
[2, 4, 5, 2 | 5 | 7, 7, 9, 8] found 5 at index 5, so this is median

(Pick 5 at index 6):
A = [5, 2, 7, 9, 4, 5, 7, 2, 8], swap pivot with last...
[5, 2, 7, 9, 4, 8, 7, 2, 5] then will swap with 5, 2, 4, and 2.
[5, 2, 4, 2, 7, 8, 7, 9, 5] then move pivot into proper place
[5, 2, 4, 2 | 5 | 8, 7, 9, 7] found 5 at index 5, so this is median

Problem 3. (30 points)
In programming assignment 1, you coded a recursive bucket sort where a constant number of buckets were chosen, and if the amount of items in a bucket was larger than another constant value then all the items in that bucket were recursively bucket sorted. What is both the average and worst case runtime for this algorithm? (You can assume there are no duplicate items.) Give the tightest bound you can find in big-O notation. Just like in the programming assignment, you can assume that we are bucket-sorting strings.

Average case:
We assume if we have K buckets that there are n/k elements per bucket. Let C be our bucket capacity (if more than C elements in bucket, we recursion). If n/K elements is larger than C, we would do recursion. This would mean originally n/K elements and K buckets. Thus on average n/K^2 elements per bucket. Every recursion adds another power of K in the denominator. Let D be the first time $n/K^D < C$. At this point, it takes a constant amount of time to sort. At each level of this recursion, we will sort all n elements into buckets, so the runtime is O(n*D).

To find D, we look at: $n/K^D < C$. This is the smallest D that makes this true, so:
$n/K^{D-1} \geq C$

Rearranging gives:
$n \geq C * K^{D-1}$ (taking logs of both sides)
$\lg(n) \geq (D-1) \lg(C*K)$
$\lg(n) \geq D \lg(C*K) \geq D \lg(C*K) - \lg(C*K)$

lg(C*K) is a constant, so D is O(lg(n)), thus the overall average run-time is O(n lg n).

Such a formal proof is not necessary, it would be sufficient to argue that the height of the recursion is lg(n) as there is are an exponential amount of buckets being created.

You could also... you know... use the master's theorem. k buckets with n/k per bucket. You need to loop through the array once (with at most k comparisons) to find the appropriate bucket. As k is a constant, this runs in O(n). This gives:

T(n) = k T(n / k) + O(n)
k T(n / k) acts like n^(log_k(k)) = n^1. Since both parts are O(n), we get a tie. This means we multiply by lg(n). Total runtime is then O(n lg n).

Worse case:
The worst case is that all the strings fall into the same bucket. However this cannot happen forever, as the strings are unique. So the worst case is the all except the very end of the strings are the same. Once again, at each level of recursion all n strings must be put into a bucket (the same one). Only on the last part of the string do they start going into different buckets. Thus the runtime is O(n*T) where T is the max length of the strings.

(Technically if T < lg n, the runtime is still O(n lg n) but it is not necessary to mention this case.)

Problem 4. (20 points)
Prove that $f(n) = n^3 - 2n + 4$ is $O(n^4)$. (Yes, that is a 4 in there.)

To prove big-O, we need to provide a c and n_0.

So we want to find:
$n^3 - 2n + 4 < c\, n^4$, for all $n > n_0$
Let's pick c=5.
$n^3 - 2n + 4 < n^3 + 4 < 5n^3$ (when $n > 1$)
Thus $5n^3 < 5n^4$ for any $n > 1$
Thus we can pick c=5 and $n_0 = 1$

Problem 5. (20 points)
You are managing a train station and have an array of train arrival and departure times, with each index representing a different train. Outline an algorithm with loose pseudo-code or a description that finds the maximum number of trains in the station at the same time (i.e. how many tracks must the station have). This algorithm must run in O(n lg n) with using only a constant amount of extra memory (i.e. it cannot be related to the number of trains arriving this day).

Example input arrays:
Arrivals = [9:10, 11:50, 8:40, 15:20, 15:20, 12:30]
Departures = [9:40, 12:40, 9:00, 15:40, 16:10, 13:00]
This means the first train will arrive at 9:10 and depart at 9:40. The second train will arrive at 11:50 and depart at 12:40.

First we sort both the arrivals and departure arrays. As we cannot use extra memory, this needs to be an in-place algorithm. The only sorts that do this are: insertion sort and heap sort (quicksort uses lg(n) memory, so only minor deductions for choosing this). As we need to maintain an O(n lg n) runtime, we must use heapsort (and not insertion sort).

Let A = sorted arrival array, D = sorted departure array.
The rest of the problem I give pseudo-code

```
MaxTrains(A, D):
i = 1    // for arrival trains
j = 1   // for departing rains
k = 0
max_trains = 0
while( i <= n )

        if( A[i] < D[j] )   // new train is arriving first
                i = i + 1
                k = k + 1
        else    // next event is a train departure
                j = j + 1
                k = k - 1

        if ( k > max_trains )
                max_trains = k

return max_trains
```

We can stop when “i” reaches the end of the arrival array as after this point, trains will only leave and we will not find a max here. We also know that “j” cannot go out of the array before “i” as the arrival times are always before the departure times.

The above pseduo-code runs in O(n) time as we need to loop through each element once (while loop). Inside the while-loop is at most 2 comparisons and 3 assignments (a constant amount of work). So the overall runtime is bounded by the sorting of A and D, which takes O(n lg n)

(Proving run-time is not necessary for points)