CSCI3104: Algorithms, Fall 2017

Midterm Exam

Name: _____

You have 120 minutes to work on the exam. Please write down your solutions clearly and with enough detail explaining or justifying your solutions.

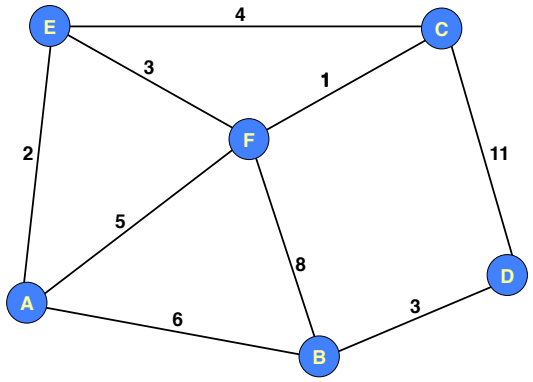
You can have one sheet of notes during the exam. Write all answers in your blue book. Your answers need to be clearly marked to receive credit.

Please put your cell phone away for the duration of the exam. If we see your cell phone during the exam, you will receive a 0 and be asked to leave the room.

1. (20 points total, 2 points for each question) For this problem, you only need to give an answer but do not need to justify the answer.
   (a) What is the asymptotic relation, Θ, O or Ω, for $f(n) = 10n \log n$ in terms of $g(n) = n^{\frac{4}{3}}$?
   Answer: $f(n) = O(g(n))$
   (b) What is the asymptotic solution to the recurrence $T(n) = 3T(\frac{n}{3}) + cn$?
   Answer: $T(n) = \Theta(n \log n)$
   (c) If you apply the Master method to solve for the recurrence $T(n) = 4T(\frac{n}{2}) + n^2$, which case of the Master Theorem applies?
   Answer: Case 2
   (d) True or False: The asymptotic solution to the recurrence $T(n) = T(\frac{n}{20}) + T(\frac{19n}{20}) + cn$ is closer to $\Theta(n^2)$ than to $\Theta(n \log n)$?
   Answer: False
   (e) For the deterministic QuickSort where you always choose the first element of an array as the pivot, what kind of input gives you the worst case performance?
   Answer: An increasing sequence, or a decreasing sequence
   (f) Given an alphabet of five symbols: a, b, c, d and e, with frequencies 0.4, 0.12, 0.2, 0.15, 0.13 respectively, what are the Huffman codes for the symbols b and c?
   Answer: Any 3-digit codes whose first digits are the same and second digits are different; e.g., $(000, 011)$, $(010, 001)$, and $(110, 100)$, etc.
   (g) Suppose we use a hash function h to hash 2000 distinct keys into a table of 1000 slots. Assuming simple uniform hashing, what is the probability that two keys are hashed to the same slot?
   Answer: 1/1000
   (h) Given the hash function $h(k) = k$ mod 11, give an example input of 8 keys that leads to the worst case performance.
   Answer: Any input where the keys have the same remainder when divided by 11
   (i) Assume you have a complete binary tree structure with weighted edges. You implement the following greedy algorithm to find the shortest distance from the root to the leaf, where shortest distance is the path with the lowest sum:
   Starting from the root, travel from each node to its child via the edge that has the least weight. Stop when you reach a leaf.
   Provide an example tree where this algorithm will not find the optimal solution.
   Answer: Any tree that violates the optimal substructure property

(j) Given the following weighted graph (see next page), if you apply the Prim's algorithm for finding a minimum spanning tree and start with node A, what is the 4th edge to be included?

Answer: Edge AB



2. (a) (5 points) Write an iterative (or recursive) algorithm to compute the product of two positive integers *m* and *n* using only addition.

   (b) (5 points) What is the Θ behavior of your algorithm?

Answer:

```
computeProduct(m, n)
    sum = 0
    for i = 1 to n {                    # loop n times
       sum += m
    }
    return sum
```

Θ(n)

3. (5 points) You are given two arrays of integers A and B, both of which are sorted in ascending order. Consider the following algorithm for checking whether or not A and B have an element in common.

```
findCommonElement(A, B) :
    # assume A,B are both sorted in ascending order
    for i = 0 to length(A) {                    # iterate through A
        for j = 0 to length(B) {                # iterate through B
            if (A[i] == B[j]) { return TRUE }   # check pairwise
        }
    }
    return FALSE
```

Provide an input for A and B that results in *findCommonElement* returning FALSE.
Answer: Any two arrays that don't share a common element.

4. Consider the following algorithm:

```
Let A = [1 ...n], where A[ i ] = i
def foo(A, s, t):
  if(t >= 1):
    mid = floor((s + t) / 2)
    add = 0
    for i = s to t:
      add += A[ i ]
    print(add)
    foo(A, 1, mid)
    foo(A, mid+1, n)
```

(a) (10 points) Write down a recurrence for the algorithm and solve it using the Master method.
Answer: $T(n) = 2T(\frac{n}{2}) + cn$
$T(n) = \Theta(n \log n)$

(b) (10 points) Which of the following sequences is a valid output of the algorithm?

i. 21, 6, 15, 6, 3, 3, ...
ii. 21, 15, 6, 9, 6, ...
iii. 6, 15, 21, 36, 21, ...

Answer: None of these answers are correct. Question not graded.

5. Consider the following QuickSort and Partition algorithms, where Partition uses the last element in the subarray as the pivot value:

```
QuickSort(A, p, r):
  if(p < r):
    q = Partition(A, p, r)
    QuickSort(A, p, q-1)
    QuickSort(A, q+1, r)


Partition(A, p, r):
  x = A[r]
  i = p-1
  for j = p to r-1:
    if A[j] <= x:
      i++
      swap(A[i], A[j])
  swap(A[i+1], A[r])
  return i+1
```

(a) (5 points) Describe the lines of code that change, and how they change, in Partition if floor((p + r) / 2) is used as the pivot value.

(b) (5 points) For the input array A = [2, 6, 4, 1, 5 3], what are the values used for the pivot to sort the array. Use floor((p + r) / 2) as the pivot.

Answer: The easiest approach is to swap the pivot value with A[r] before the start of the for loop, and then set x = A[r]. If you do that, no other code has to change. The pivots are 2, 3, 4, 5.

6. Consider the following algorithm called Selection Sort, which sorts an array in ascending order:

```
SelectionSort(A):
 int i,j
 int n
 for (j = 0; j < n-1; j++)
   int iMin = j
   for (i = j+1; i < n; i++)
     if (A[i] < A[iMin])
      iMin = i
   if (iMin != j)
    swap(A[j], A[iMin]);
```

(a) (5 points) What is the Θ, Ω, O of Selection Sort?
Solution: $\Theta(n^2), \Omega(n^2), O(n^2)$.

(b) (5 points) What is the loop invariant of Selection Sort?
Solution: At the beginning of each iteration ($j th$) of the outer loop, the subarray $A[0, \cdots, j-2]$ is sorted and contains the $j-1$ smallest elements of the original array.

(c) (10 points) Use the loop invariant to show the correctness of the Selection Sort algorithm

7. (10 points) Given two lists of integers $\{a_1, a_2, \ldots, a_n\}$ and $\{b_1, b_2, \ldots, b_n\}$, we want to determine whether there exist $i,\ j$ such that $a_i + b_j = 0$. Show how to solve this problem using $O(n \log n)$ operations.

   Solution: The solution may not be unique. Here is one: First, use merge sort to sort the sequence $b = \{b_1, b_2, \ldots, b_n\}$, which will take $O(n \log n)$ time. Then, for each element a_i, use binary search to see if $-a_i$ is in the sorted sequence b. Each binary search takes $O(\log n)$ time, and n binary searches will take $O(n \log n)$ time. The total running is still $O(n \log n)$.

8. Given a list of integers $A = \{a_1, a_2, \ldots, a_n\}$, we want to find a subsequence having maximum sum; i.e., if for $i \leq j$ we define $A_{i,j} = \sum_{i \leq k \leq j} a_k$, we want i, j such that $A_{i,j}$ is maximum. Consider the following divide-and-conquer algorithm:

```
MaxSumSDC(A, s, t) {
    if (s  > t) { return 0 }
    if (s == t) { return A[s] }

    m = floor( (s + t)/2 )

    leftMax = sum = 0
    for (i = m, i >= s, i--) {
            sum += A[i]
            if (sum >= leftMax) { leftMax = sum }
    }

    rightMax = sum = 0
    for (i = m+1, i <= t, i++) {
            sum += A[i]
            if (sum > rightMax) { rightMax = sum }
    }

    spanMax = leftMax + rightMax
    halfMax = max( MaxSumSDC(A, s, m),  MaxSumSDC(A, m+1, t) )
    return max(spanMax, halfMax)
}
```

To find the maximum sum of subsequence of A, you call $MaxSumSDC(A, 1, n)$. Let $T(n)$ be the running time of the above algorithm on an input of size n.

(a) (10 points) Write down the recurrence relation for $T(n)$ including the base case.
Solution: $T(n) = 2T(\frac{n}{2}) + \Theta(n)$ and base case $T(1) = c$.

(b) (5 points) Solve the recurrence relation to obtain the asymptotic running time.
Solution: Using either recursion tree approach or master approach yields $T(n) = \Theta(n \log n)$.