ANSWER KEY

CMPS 102 Midterm, Winter 2019

This is a 65 minute, timed, closed book exam (although you may have a 3-by-5 card of handwritten notes, one side). Put your name, SID and account at the top of each page. Write your answers on the exam itself. No calculators, phones, or other electronic devices. Show your student ID as you turn in your exam. You may use the back sides of pages 1 and 2 as scratch space. Standard test taking advice applies: read the questions carefully.

Good Luck

problem	points	score
1	16	
2	20	
3	24	
total	60	

ANSWER KEY

Problem 1: For the first two **circle** the best of the colon (":") separated alternatives.

- (1 pts) In the stable matching algorithm presented in class, each hospital proposed to students who either tentatively accepted the proposal or rejected it if they already had a proposal from a hospital they preferred. This algorithm always produced the best feasible outcomes for [✓ *hospitals*] *: students : both : neither.*

- (1 pts) The best known asymptotic worst-case time for multiplying two $n \times n$ matrices is $O(n^c)$ where: $c < 2$: $c = 2$: [✓ $2 < c < 2.5$] : $c = 2.5$: $2.5 < c < 3$: $c = 3$.

- (4 pts) Give the asymptotic growth rates of the functions defined by the following recurrence relations, no proof or justification required. Assume the boundary conditions are small positive values.
  - _________ $\Theta(n \log n)$ $T(n) = 2T(n/2) + 3n$ note: $O()$ also OK
  - _________ $\Theta(n)$ $T(n) = T(n/2) + 2n$
  - _________ $\Theta(n)$ $T(n) = T(n/2) + T(n/3) + n$
  - _________ $\Theta(n^{\lg 3})$ $T(n) = 3T(n/2) + 1$

- (2 pts) What is the worst-case running time of DFS on an n-vertex, m-edge graph:

  When the graph is represented using adjacency lists? ___________ $O(n+m)$ or $O(m)$

  When the graph is represented using an adjacency array? ___________ $O(n^2)$

- (3 pts) The FFT algorithm presented in class finds a point-value representation of an $n = 2^k$ coefficient polynomial P by combining the point-value representation of two "smaller" polynomials, P_{even} and P_{odd}. What complex-valued points are used for:

  the representation of P_{even}? The $n/2$th complex roots of 1

  the representation of P_{odd}? The $n/2$th complex roots of 1

  for the representation of P? The nth complex roots of 1

- (5 pts) Dijkstra's and Prim's algorithms use of a priority queue. What priority queue operations do they use? What is the worst-case running time (as a function of n, the number of elements stored) of these operations assuming a binary heap implementation?

insert(item, value)	$O(\lg n)$
decrease(item, newvalue)	$O(\lg n)$
extract_min()	$O(\lg n)$

  May have remove(item) or delete(item) instead of decrease(...), and may have inqueue(key).

Name: ______________________________ SID: _________ Acct: _________@ucsc.edu

Problem 2: divide and conquer (20 pts)

Consider the problem of given an array $A[0..n-1]$ of n distinct integers (so no integer occurs twice) either find an index i such that $A[i] = i$ or report "no such index".

For example, if $A = [-3, 0, 2, 3, 7]$ then either 2 or 3 are correct answers, but if $A = [-2, -1, 8, 10]$ then the correct answer is "no such index".

(A) Clearly describe a divide and conquer algorithm that correctly solves this problem in $O(\log n)$ time (7 pts).
(B) Prove that your algorithm is correct (11 pts).
(C) Write the recurrence for your algorithm's running time (2 pts).
(Possible hint: since the integers are distinct and sorted, each $A[i+1] \geq A[i] + 1$.)

Solution to (A): return FindPlace$(A, 0, n-1)$.

```
function FindPlace($A, \ell, u$ )
    if $\ell > u$ then
        return "no such index"
    end if
    if $\ell = u$ then
        if $A[\ell] = \ell$ then
            return $\ell$
        else
            return "no such index"
        end if
    end if
    $m \leftarrow (\ell + u)/2$                              ▷ round down so $\ell \leq m < u$
    if $A[m] = m$ then
        return $m$
    else if $A[m] < m$ then
        return FindPlace$(A, m+1, u)$
    else
        return FindPlace$(A, \ell, m)$
    end if
end function
```

Solution to (B).
From the hint we have the following:
Fact 1: if at some index i, $A[i] < i$ then for all indices $0 < j < i$ we have $A[j] < j$.
Fact 2: if at some index i, $A[i] > i$ then for all indices $i < j < n-1$ we have $A[j] > j$.

Claim 1. *Function findPlace(A, ℓ, u) returns a index $i \in \{\ell, \ell+1, \ldots, u\}$ where $A[i] = i$ if there is one, and returns "no index found" if for all indices $i \in \{\ell, \ell+1, \ldots, u\}$, $A[i] \neq i$.*

Proof. By induction on $u - \ell$. Let $P(k)$ be the property that for any sorted array n of distinct integers, and any $0 \leq \ell$ and $u = \ell + k \leq n - 1$, findPlace(A, ℓ, u) correctly returns an appropriate index of the "no index found" message.
Base Case: $k = 0$ (so $u = \ell$), prove $P(0)$.
When $k = 0$ we have $\ell = u$ and the **if** statement at line 5: will be executed. This returns ℓ if $A[\ell] = \ell$ and "no index found" when $A[\ell] \neq \ell$. This shows $P(0)$.
Base case: $k = 1$ (so $u = \ell + 1$) , prove $P(1)$.
In this case $m = \ell$. If $A[\ell] = \ell$ then the value $m = \ell$ will be returned (line 14).
If $A[\ell] < \ell$ then the algorithm returns the result of calling $A[u, u]$, which is u if $A[u] = u$ and "no index found" otherwise. If $A[\ell] > \ell$ then the algorithm again calls FindPlace(A, ℓ, ℓ) and will return "no index found". This is correct since $A[\ell + 1] > \ell + 1$ also, so $A[u] \neq u$.

Inductive step: Assume $k > 1$ and $P(j)$ for all $0 \leq j < k$ to show that $P(k)$ is also true.
When $k > 1$, we have $u \geq \ell + 2$, so $\ell < m < k$.

If $A[m] = m$ then the algorithm returns the correct answer m on line 14.
If $A[m] < m$ then for each $i \leq m$ we have $A[i] \leq i$, so no index $i \in \{\ell, \ell + 1, \ldots, m\}$ has the property that $A[i] = i$, and the only values in $A[\ell..u]$ that can equal their index are stored in indices above m. In this case the algorithm calls FindPlace$(A, m + 1, u)$ which, by the inductive hypotheses, returns either an index i where $A[i] = i$ or "no index found" if no such $i \in \{m + 1, \ldots, u\}$ exists, and in both cases this is a correct answer.
If $A[m] > m$ then $A[i] > i$ for all $i \in \{m, \ldots, u\}$ so the only values in $A[\ell..u]$ that can be equal to their index are in $A[\ell..m-1]$. By the induction hypothesis, FindPlace(A, ℓ, m) finds such a place if one exists and returns "no index found" otherwise, so the correct value is computed.
This establishes $P(k)$ as desired, completing the induction. □

Solution to (C):
The recurrence is $T(n) = T(n/2) + 1$ and the boundary condition is $T(1) = 1$. (Boundary condition probably not needed)

Problem 3: greedy method. (24 pts)

Consider a long country road with n houses located on it. The start (west end) of the road is at mile 0, and the array $h[0..n-1]$ contains the locations of the houses in (real-valued) miles from the start of the road. Assume that the array h is sorted in increasing order.

In order to ensure good cell service at a house, there must be a cell tower located within 4 miles of the house. The problem is to determine locations (in terms of miles from the start) for cell towers such that both:
(1) every house is within 4 miles of a cell tower, and
(2) subject to (1), as few as possible cell towers are used.
Assume that cell towers can be located any place along the road.

For example, if there were 5 houses at locations 26, 30.2, 34, 52.3, and 53.6 then placing cell towers at locations 30 and 55.1 is a possible solution.

(A) Clearly describe a greedy algorithm solving this problem (6 pts)
(B) Argue (informally) that your algorithm places a cell tower within 4 miles of every house (6 pts)
(C) Prove that your algorithm always uses as few cell towers as possible (12 pts)

Solution to (A):

```
1: mark all houses as uncovered
2: repeat
3:     get the first uncovered house location (closest to mile 0)
4:     place a cell tower at the location 4 miles past this first uncovered house
5:     mark all houses within 4 miles of the new cell tower as covered
6: until all houses covered
```

Solution to (B):
The algorithm covers at least one house each iteration of the loop, so eventually all houses will be covered and the loop will terminate. A house gets marked as covered only when a cell tower is placed within 4 miles of it, so when all houses are marked as covered there will be a cell tower within 4 miles of every house.

Solution to (C):
Let $G_1 < \cdots < G_k$ be the milages where the greedy algorithm places cell towers.

Claim 2. *Let S be an arbitrary solution that places cell towers at locations $S_1 < S_2 < \cdots < S_\ell$. The greedy algorithm stays ahead of , $G_i \geq S_i$ for all $1 \leq i \leq \min(k, \ell)$.*

Proof. By induction on i, let $P(i)$ be the proposition that $G_i \geq S_i$.
Base Case: show $P(1)$.
S_1 must be within 4 miles of the first house in order for it to be covered. G_1 is exactly 4 miles past the first house. Therefore $S_1 \leq G_1$.
Inductive Step: Assume $i > 1$ and $P(i-1)$ is true to show $P(i)$ is true.
Let h be the location of the first house more than 4 miles past G_{i-1}. We have $S_{i-1} \leq G_{i-1}$ by $P(i)$, so h is also more than 4 miles past S_{i-1}. Some cell tower S_j for $j \geq i$ must cover

house h, so S_i must be before $h+4$ miles (Either S_i covers house h, or S_i is before the S_j that covers house h). By definition of the algorithm, G_i is at location $h+4$, and thus $S_i \leq G_i$. □

Claim 3. *Every optimal schedule uses at least k cell towers, so the greedy algorithm is optimal.*

Proof. Assume to the contrary that some schedule S places cell towers at locations $S_1 < S_2 < \cdots < S_\ell$ where $\ell < k$. Since the greedy algorithm uses another cell tower, there is a house more than 4 miles past G_ℓ. By the previous claim, $S_\ell \leq G_\ell$, so that house is also more than 4 miles past S_ℓ. Since S puts no cell tower is past S_ℓ, that house is not within 4 miles of S's cell towers, and S violates criteria (1), contradicting the assumption that S was optimal. □