CS570 Fall 2017: Analysis of Algorithms Exam III

	Points
Problem 1	20
Problem 2	15
Problem 3	10
Problem 4	15
Problem 5	15
Problem 6	10
Problem 7	15
Total	100

Instructions:

1. This is a 2-hr exam. Closed book and notes
2. If a description to an algorithm or a proof is required please limit your description or proof to within 150 words, preferably not exceeding the space allotted for that question.
3. No space other than the pages in the exam booklet will be scanned for grading.
4. If you require an additional page for a question, you can use the extra page provided within this booklet. However please indicate clearly that you are continuing the solution on the additional page.
5. Do not detach any sheets from the booklet. Detached sheets will not be scanned.
6. If using a pencil to write the answers, make sure you apply enough pressure so your answers are readable in the scanned copy of your exam.
7. Do not write your answers in cursive scripts.

1) 20 pts

Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification.

[TRUE/[✓ FALSE]]

To prove that a problem X is NP-complete, it is sufficient to prove that 3SAT is polynomial time reducible to X.

[TRUE/[✓ FALSE]]

Finding the minimum element in a binary max heap of n elements takes $O(\log n)$ time

[TRUE/[✓ FALSE]]

We are told that in the worst case, algorithm A runs in $O(n \log n)$ and algorithm B runs in $O(n^2)$. Based on these facts, there must be some N that when n>N, algorithm A runs faster than algorithm B.

[[✓ TRUE]/FALSE]

The following recurrence equation $T(n)=3T(n/3) + 0.1\, n$ has the solution:

$T(n)= \Theta\ (n \log(n^2))$.

[[✓ TRUE]/FALSE]

Every problem in NP can be solved in exponential time by a deterministic Turing machine

[[✓ TRUE]/FALSE]

In Kruskal's MST algorithm, if we choose edges in decreasing (instead of increasing) order of cost, we will end up with a spanning tree of maximum total cost

[TRUE/[✓ FALSE]]

If all edges in a graph have capacity 1, then Ford-Fulkerson runs in linear time.

[TRUE/[✓ FALSE]]

If problem X can be solved using dynamic programming, then X belongs to P.

[[✓ TRUE]/FALSE]

If Vertex-Cover $\in$ P then SAT $\in$ P.

[TRUE/[✓ FALSE]]

Assuming P!=NP, and X is a problem belonging to class NP. There is no polynomial time algorithm for X.

2) 15 pts

We have a directed graph G=(V, E) where each edge (u,v) has a length l(u,v) that is a positive integer. Let n denote the number of vertices in G. Suppose we have previously computed a n x n matrix d[·,·], where for each pair of vertices u,v ∈V, d[u,v] stores the length of the shortest path from u to v in G. The d[·,·] matrix is provided for you.

Now we add a single edge (a,b) to get the graph G' = (V,E'), where E' = E ∪{(a,b)}. Let l(a,b) denote the length of the new edge. Your job is to compute a new distance matrix d'[·,·], which should be filled in so that d'[u,v] holds the length of the shortest path from u to v in G', for each u,v ∈V.

(a) Write a concise and efficient algorithm to fill in the d'[·,·] matrix. You should not need more than about 3 lines of pseudocode. (12 pts)

Solution:

```
For each pair of vertices u,v ∈V:
        Set d'[u,v] = min(d[u,v],d[u,a] + l(a,b) + d[b,v]).
```

Grading:

- Considering all pairs of vertices (6 points)
- Comparing the current shortest path with the length of the path going through edge (a,b) and choosing the smaller path (6 points)

(b) What is the asymptotic worst-case running time of your algorithm? Express your answer in O(·) notation, as a function of n and |E|. (3 pts)

Solution:

$O(n^2)$.

Grading:

- Correct Answer (3 points)
- Incorrect answer (0 points)

3) 10 pts.

Recall the 0/1 knapsack problem:

Input: n items, where item i has profit p_i and weight w_i, and knapsack size is W.
Output: A subset of the items that maximizes profit such that the total weight of the set $\leq$ W.
You may also assume that all $p_i \geq 0$, and $0 < w_i \leq W$.

We have created the following greedy approximation algorithm for 0/1 knapsack:

Sort items according to decreasing p_i/w_i (breaking ties arbitrarily).
While we can add more items to the knapsack, pick items in this order.

Show that this approximation algorithm has no nonzero constant approximation ratio. In other words, if the value of the optimal solution is P*, prove that there is no constant ρ ($1 > \rho > 0$), where we can guarantee our greedy algorithm to achieve an approximate solution with total profit ρ P*.

Solution:
The definition of a constant factor is that P >= ρ P*. Giving an example were P = P* and saying ρ=*1* or giving two different examples and showing the ratio P/P* is not always the same does not mean there does not exist a constant approximation ratio.

The correct way to show this is not possible is to provide an example with multiple (at least two) items and their respective weights and profits. Then you need to show that the greedy algorithm and the optimal algorithm will generate two different solutions (/profits) and need to show in your example the ratio P/P* can be unboundedly small.

One example can be as follows:
Consider an item with size 1 and profit 2, and another with size W and profit W. Our greedy algorithm above will take the first item, while the optimal solution is taking the second item, so this algorithm has approximation ratio 2/W which can go arbitrarily close to zero.

Grading:
A correct solution has 10 points.

Common Mistakes:

- Giving two different examples and showing that the ratio of greedy to optimal is different doesn't mean there is no constant approximation ratio.
- The existence of a constant approximation factor will not imply P = NP.
- Greedy is not optimal!!

4) 15 pts.

The graph five-coloring problem is stated as follows: Determine if the vertices of G can be colored using 5 colors such that no two adjacent vertices share the same color.

Prove that the five-coloring problem is NP-complete.

Hint: You can assume that graph 3-coloring is NP-complete

Solution:

Show that 5-coloring is in NP:
Certificate: a color solution for the network, i.e., each node has a color.
Certifier:

i. Check for each edge (u,v), the color of node u is different from the color of node v;
ii. Check at most 5 colors are used

This process takes O(m+n) time.

Show that 5-coloring is NP-hard: prove that 3-coloring $\leq_p$ 5-coloring

Graph construction:

Given an arbitrary graph G. Construct G' by adding 2 new nodes u and v to G. Connect u and v to all nodes that existed in G, and to each other.

G' can be colored with 5 colors iff G can be colored with 3 colors.

i. If there is valid 3-color solution for G, say using colors {1,2,3}, we want to show there is a valid 5-coloring solution to G'. We can color G' using five colors by assigning colors to G according to the 3-color solution, and then color node u and v by additional two different colors. In this case, node u and v have different colors from all the other nodes in G', and together with the 3-coloring solution in G, we use at most 5 colors to color G'.
ii. If there is a valid 5-coloring solution for G', we want to show there is a valid 3-coloring solution in G. In G', since node u and v connect to all the other nodes in G and to each other, the 5-coloring solution must assign two different colors to node u and v, say colors 4 and 5. Then the remaining three colors {1,2,3} are used to color the remaining graph G and form a valid 3-color solution.

Solving 5-coloring to G' is as hard as solving 3-color to G, then 5-coloring problem is at least as hard as 3-coloring, i.e., 3-coloring $\leq_p$ 5-coloring.

Grading Rubrics:

1. Prove the problem is NP (3 points in total)
   - i) Mentioning the necessity of proving NP (1 point)
   - ii) Checking two connecting nodes (u,v) having different colors (1 point)
   - iii) Checking the total number of colors used is at most (1 point)
2. Prove the problem is NP-hard (12 points in total)
   - i) Graph construction (4 points in total)
     - a. Adding two additional nodes (1 point)
     - b. Connecting to all the nodes in G (2 point)
     - c. Connecting node u and v (1 point)
   - ii) Proof (8 points in total)
     - a. 3color -> 5 color (2 points). This is trivial even under other constructions (even incorrect constructions)
     - b. 5color -> 3 color (6 points).
       - If your graph construction is incorrect, you will get 0 point in this part.
       - Under the graph construction posted in the standard solution:
         - ✓ Explaining the two additional nodes u and v must have colors different from those used in G (3 points)
         - ✓ Explaining the two additional nodes u and v must have two different colors in between (3 points)

Popular Mistakes:

1. Try to show, if G is 3-colorable, then G is 5-colorable
   - – Loss 4 points for construction (actually no construction is given);
   - – Lose 6 points for proof (proof for 5 color-> 3 color cannot be correct)
2. Without adding additional nodes to G, but try to “manipulate colors” during construction
   - – Loss 4 points for construction (actually no construction is given);
   - – Lose 6 points for proof (proof for 5 color-> 3 color cannot be correct)
3. In the graph construction, someone added additional nodes to G, but assigned additional colors to during the construction, and then in the proof, claim them “using two different colors”.
   - – Loss 3 points for construction (wrong construction);
   - – Loss 6 points for proof
4. In the graph construction, for each node i in G, someone added two additional nodes ui and vi while connecting them to node i, and then connect or not connect ui and vi.
   - – Loss 2 points for construction (at least they know connecting with somewhere in G);
   - – Loss 6 points in proof
5. In the graph construction, someone added two nodes u and v to G and connect them to all the nodes in G, but did not connect them to each other
   - – Loss 1 point for construction

- If they can correctly explain that, to prove G has 3-color solution given G' having 5-color solution, their construction can guarantee node u and v take different colors from all the nodes in G, they will only lose 3 points; otherwise, they will lost 6 points.

6. In the graph construction, someone added two nodes "into" each edge
   - Loss 2 points for construction
   - Loss 6 points in proof

5) 15 pts.

The price of the recently introduced cryptocurrency, Crypcoin, has been changing rapidly over the last two months. Given the hourly price chart of this currency for the last two months, you are tasked to find out the maximum profit one could have made by buying 100 Crypcoins in one transaction and subsequently selling 100 Crypcoins in one transaction within this period.

Specifically, given the price chart of Crypcoin over this period P[i] for i = 1 to n, we're looking for the maximum value of (P[k]−P[j])*100 for some indices j, k, where $1 \leq j < k \leq n$.

Examples: If the array is [2, 3, 10, 6, 4, 8, 1] then the returned value should be 8 *100 (Diff between 2 and 10, times 100 Cyrpcoins). If the array is [7, 9, 5, 6, 3, 2] then the returned value should be 2 * 100 (Diff between 7 and 9, times 100 Cyrpcoins)

Describe a divide and conquer algorithm to solve the problem in O(n log n) time. You do not need to prove that your algorithm is correct.

Solution:

```
CrypMAX(P[1..n]):
If $n \leq 1$, return 0.
Set $max_L$ := CrypMAX(P[1,… , $\lfloor n/2 \rfloor$])
Set $max_R$ := CrypMAX(P[$\lfloor n/2 + 1 \rfloor$,…,n])
Set x := min(P[1, … , $\lfloor n/2 \rfloor$]).
Set y := max(P[ $\lfloor n/2 + 1 \rfloor$,…,n]).
Return max($max_L$,$max_R$, (y−x)*100 ).
```

Complexity Analysis: There are two recursive calls on a subarray of half the size, plus a linear ($\Theta(n)$) scan to find the smallest/largest of either half. Thus, the recurrence relation is $T(n) = 2T(n/2) + \Theta(n)$, which solves to $\Theta(n \log n)$.

Alternate solution:

We can modify the algorithm to also return the smallest and largest elements of the array (i.e. return the triple (best price, min of array, max of array)). This allows the algorithm to avoid the computation of finding the smallest/largest elements of each half of the array, as the recursive calls have already calculated them. (As an additional detail, the base case also needed a minor tweak.) Our runtime would thus be $T(n) = 2T(n/2) + \Theta(1)$, which solves to $\Theta(n)$.

Grading rubric:

1. Basic structure for an efficient divide-and-conquer algorithm (5 pts)
   a. No ambiguity, e.g., "divide it into several subproblems" or "divide it into two subproblems" will not receive the 5 pts. "… two halves" or "… two subproblems with equal size" are acceptable.
2. Correctly conquer the problem by finding the cross-segment term: max(P(mid:end)-min(P(1:mid). (5 pts)
   a. The max/min operation are required to be written explicitly, to achieve an O(n) complexity for each step. Finding the maximal different between two halves in brute force requires O(n^2).
3. Complete the entire algorithm (5 pts). Examples:
   a. -1 pts for missing base case;
   b. -1 pts for multiplying by 100 more than once inappropriately.
4. Non-divide-and-conquer algorithms:
   a. Correct O(n log(n)) algorithm receive 10 pts.
   b. Correct O(n^2) algorithm receive 5 pts.
   c. Other receive 0 pt.

Note and common mistakes:

1. You cannot buy the coin after your sell. (sell must occur no earlier than buy).
   a. You cannot simply pick the highest price to sell and the lowest price to buy.
2. If you buy on i-th day, you don't have to sell it on the i+1-th day.
3. You cannot do binary search on unsorted array.

4. [Most common mistake] The buy and sell prices does not have to be the lowest or highest price. For example, [10,4,5,1]. **Therefore, returning the best buy/sell dates from the subproblems does not help you with merging the two subproblems, which requires the highest/lowest price from them.**

Partially correct example solutions and grades:

1. Brute force search, O(n^2) (5pts)

```
Ret = 0;
For i in 1 to n:
    For j in i+1 to n:
        Ret = max(Ret, P(j)-P(i));
Return 100*Ret;
```

Note: the recursive version: F(i,j) -> f(i+1,j) and f(i, j-1) and P(j)-P(i) also receive 5 pts.

2. Non-divide-and-conquer O(n) (10 pts)
   Segment P into fewest monotonical increasing intervals. Calucate the max-difference within these intervals and return the largest difference * 100.
3. Non-divide-and-conquer One-pass O(n) (10 pts)

```
Min = P(1), Ret = 0;
```

```
For i in 1 to n:
    Ret = max(Ret, P(i)-Min)
    Min = min(Min, P(i))
Return Ret*100
```

4. Divide-and-conquer O(n^2) (5 pts)

```
Def Solve(P[1:m]):
k = argmax(P)
Return max(P(k)-min(P[1:k]), Solve(P[k+1:m])
```

Final solution is 100*Solve(P); Note argmax is O(n) operation on unsorted array.

6) 10 pts.

Recall the Circulation with Lower Bounds problem:

- Each directed edge e has a lower bound l_e and an upper bound c_e on flow
- Node v has demand value d_v, where $d_v$could be positive, negative, or zero
- Objective is to find a feasible circulation that meets capacity constraints over each edge and satisfies demand conditions for all nodes

We now add the following constraints and objective to this problem:

- For each node v with demand value $d_v = 0$, the flow value through that node should be greater than l_v and less than c_v
- Since we suspect there may be some issues at a specific node *x* in the network, we want to minimize the flow going through this node as much as possible.

Give a linear programming formulation for this problem.

Solution:

Objective: minimize ∑)** (+,-. / '((4 points)

Constraints:

1. !" ≤ $' ≤ %" $'( )*%ℎ),-)) (2points)
2. !"# = ∑()) ' *+,- "#& - ∑()) ' -/, -0 "#& &12345ℎ 7328397* (2 points)
3. !"# ≤ ∑()) ' *+,- "#& ≤ ."# &/0 12.ℎ 4/51 67 89.ℎ :ℎ 2: 5"# = 0 (2 points)

- 1 point may be deducted from each of your constraints where you have not specified what is the flow
- Up to 3 points may be deducted from your objective if you have not specified what is the flow in your solution. (not defining fin for instance)

7) 15 pts

Woody will cut a given log of wood, at any place you choose, for a price equal to the length of the given log. Suppose you have a log of length L, marked to be cut in n different locations labeled $1, 2, \ldots, n$. For simplicity, let indices 0 and n + 1 denote the left and right endpoints of the original log of length L. Let di denote the distance of mark i from the left end of the log, and assume that $0 = d_0 < d_1 < d_2 < \ldots < d_n < d_{n+1} = L$. The wood-cutting problem is the problem of determining the sequence of cuts to the log that will cut the log at all the marked places and minimize Woody's total payment. Remember that for a single cut on a given log, Woody gets paid an amount equal to the length of that log. Give a dynamic programming algorithm to solve this problem.

a) Define (in plain English) subproblems to be solved. (4 pts)

Let $c(i, j)$ be the min cost of cutting a log with left endpoint i and right endpoint j at all its marked locations.

Grading:
Incomplete definition: -2 pts
No definition: -4pts

b) Write the recurrence relation for subproblems. (7 pts)

(Dynamic programming)
$c(i, j) = \min_{i<k<j}\{c(i, k) + c(k, j) + d_j - d_i\}$
Base case: if $j = i+1$, $c(i, j) = 0$

Grading:
Error in recursion: -2pts, if multiple errors, deduction adds up.
Wrong recurrence relation: -7pts
Base case is missing (case of $j = i+1$): -2pts
Wrong base case: -2pts

c) Compute the runtime of the algorithm in terms of *n*. (4 pts)

$O(n^3)$

Grading:
Used big Theta notation instead of big O notation: -2pts
Missing big O notation: -2pts
Wrong runtime complexity: -4pts