Problem samples in previous Algorithm exams

(PS: These questions are for informational purposes only and will not appear in this year's exam.)

Score	Section I: Fill in the blank

1. Finding a smallest vertex cover in an undirected tree $T = <V, E>$ can be finished in $O(|V|)$ time .

2. The hallmarks of divide and conquer are:
Optimal substructure and independent sub-problem.

3. The time complexity of the Dijkstra's algorithm is $O\left((|V|\cdot d+|E|)\frac{\log|V|}{\log d}\right)$ if the priority queue is implemented by a *d-ary* heap (d>2).

4. If $T(n)=T(n/4)+2T(n/2)+n^2$, then $T(n)$ = $O(n^2)$.

5. $\sum_{i=1}^{n} i^k$ **=** $\theta(n^{k+1})$**.**

6. The hallmarks of dynamic programming are:
optimal substructure and overlapping sub-problem.

7. Suppose edges X are part of a minimum spanning tree of G=(V, E). Pick any subset of nodes S for which X does not cross between S and V-S, and let e be the lightest edge across this partition. Then X∪{e} is a part of some MST. This is called cut property.

8. A set of nodes S is a vertex cover of graph G = (V, E) (S touches every edge in E) if and only if the remaining nodes, V-S, are an independent set of G.

Score	Evaluator	Section II: Choice Choose the most appropriate answer from each group.

1. The time complexity of finding single-source shortest paths in a directed acyclic graph (DAG) is (C).
(A) $O((|V|+|E|)\log|V|)$ (B) $O(|V||E|)$ (C) $O(|V|+|E|)$ (D) $O((|V|^3)$

2. In which of the following problems, the greedy algorithm cannot guarantee to find optimal solutions? (C)

(A) Knapsack **(B) Minimum Spanning Tree**

(C) Set Cover **(D) Huffman Encoding**

3. The running time of the Dijkstra's algorithm is (A) if the priority queue is implemented by a *binary* heap.

(A) $O\big((|V|+|E|)\log|V|\big)$ **(B)** $O(|V|\log|V|+|E|)$

(C) $O(\log d\,|V|^2)$ **(D)** $O\left((|V|\cdot d+|E|)\dfrac{\log|V|}{\log d}\right)$

4. Which of the following statements is NOT correct? (B)

(A) If each edge weight is increased by 1, the minimal spanning trees will not change.

(B) A subset $C\subseteq V$ is a vertex cover in G if and only if V-C is a clique in G.

(C) The time complexity of the Bellman-Ford's algorithm is $O(|V|\cdot|E|)$.

(D) If implemented with disjoint sets, the running time of the Kruskal's algorithm is $O((|V|+|E|)\log|V|)$.

5. If $T(n)=aT(\lceil n/b\rceil)+O(n^d)$ **for some constants a>0, b>1, and d≥0, then $T(n)$ = (D)**

if $d<\log_b a$ **?**

(A) $O(n^d)$ **(B)** $O(n^d\log_b n)$ **(C)** $O(n^d\log_2 n)$ **(D)** $O(n^{\log_b a})$

6. The least running time of algorithm which finds single-source shortest paths in a directed acyclic graph (DAG) is (C).

(A) $O((|V|+|E|)\log|V|)$ **(B) $O(|V||E|)$** **(C) $O(|V|+|E|)$** **(D) $O((|V|^3)$**

7. Which of the following problems is NOT a NP-Complete problem? (B).

(A) SAT **(B) Minimum Spanning Tree**

(C) Set Cover **(D) Independent Set**

8. The running time of the Dijkstra's algorithm is (D) if the priority queue is implemented by a *d-ary* heap (d>2).

(A) $O\big((|V|+|E|)\log|V|\big)$ **(B)** $O(|V|\log|V|+|E|)$

(C) $O(\log d\,|V|^2)$ **(D)** $O\left((|V|\cdot d+|E|)\dfrac{\log|V|}{\log d}\right)$

9. Which of the following statements is NOT correct? (A)

(A) $n!=O(2^n)$ **(B)** $10\log n=\Theta(\log(n^2))$

(C) $n^{1/2}=\Omega((\log n)^3)$ **(D)** $n^{1/2}=O(n^{2/3})$

10. Finding a smallest vertex cover in an undirected tree T = <V, E> can be finished in time (C).

(A) $O(|V|^2)$ (B) $O(|V|\cdot\log|V|)$ (C) $O(|V|)$ (D) $O(|E|^2\cdot|V|)$

11. Which of the following problems is a NP-Complete problem? (A).

(A) 3SAT (B) Maximum Spanning Tree

(C) Hoffman Encoding (D) Single-source shortest path

12. If $T(n) = 49T(n/25) + n^{3/2}\log n$, then $T(n)$ = (D)?

(A) $O(\log n)$ **(B)** $O(n^{\log_{25}49})$ **(C)** $O(n^{2/3}\log n)$ **(D)** $O(n^{3/2}\log n)$

Score	Evaluator	Section III: True/False
		Write T or F before each question.

T 1. $\sum_{i=1}^{n}\frac{1}{i}=\theta(\log n)$.

F 2. Back edges lead from a node to a non-child descendant in the DFS tree.

T 3. For the vertex cover problem, the approximation algorithm based on maximal matching has a constant approximation ratio.

F 4.The hallmarks of dynamic programming are optimal substructure and independent sub-problem.

T 5. Any problem in NP can be reduced into a 3SAT problem in the polynomial time.

T 6. A subset $S \subseteq V$ is a vertex cover in G if and only if $V-S$ is an independent set in G.

F 7.The time complexity of the Bellman-Ford's algorithm is $O((|V|+|E|)\log|V|)$.

F 8. If each edge weight is increased by 1, the shortest path will not change.

T 9. Any problem in NP can be reduced into a 3D-matching problem in the polynomial time.

T 10. Matroids exhibit the optimal-substructure property and the greedy-choice property.

Score	Evaluator	Section IV: Short Questions

1. Solve the following recurrence relations and give a θ bound for each of them.

$(1) T(n) = 9T(n/3)+n$

$$(2)\begin{cases} T(n) = T\left(\frac{n}{2}\right)+T\left(\frac{n}{4}\right)+cn \\ T(1) = 1 \end{cases}$$

$(3) T(n) = T(n-1)+\log n$

Answer:

(1) According to the master theorem:

***a*=9, *b*=3, *f*(*n*)=*n*, $\log_3 9>1$，so** $T(n) = O(n^{\log_3 9}) = O(n^2)$

(2) Use the recursive tree method:

$$T(n) = cn + \frac{3cn}{4} + \left(\frac{3}{4}\right)^2 cn + \ldots = \left[1 + \left(\frac{3}{4}\right) + \left(\frac{3}{4}\right)^2 + \ldots\right] cn = \theta(n)$$

(3)Use the iterative method:

$$T(n) = \log n + \log(n-1) + \ldots + \log 2 + T(1) = \log(n!) + T(1) = \theta(n \log n)$$

2. What are the characteristics of an algorithm?

Answer:

An algorithm is a sequence of steps which is used to solve a category of problems. Characteristics are given by:

a) **Unambiguous: every step is deterministic;**

b) **Mechanical: machine can "understand";**

c) **Finite: can be implemented in limited steps;**

d) **Input/output: to state the problem size and the result.**

3. Show that the master theorem is correct.

Master theorem If $T(n) = aT(\lceil n/b \rceil) + O(n^d)$ for some constants $a>0$, $b>1$, and $d \geq 0$, then

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log_b n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

Proof.

- Assume n is a power of b
- The total work done at the kth level

$$a^k \times O(\frac{n}{b^k})^d = O(n^d) \times (\frac{a}{b^d})^k$$

- As k goes from 0 to $\log_b n$, these numbers form a geometric series with ratio a/b^d.

1. *The ratio is less than* 1.
   Then the series is decreasing, and its sum is just given by its first term, $O(n^d)$.
2. *The ratio is greater than* 1.
   The series is increasing and its sum is given by its last term, $O(n^{\log_b a})$:

$$n^d \left(\frac{a}{b^d}\right)^{\log_b n} = n^d \left(\frac{a^{\log_b n}}{(b^{\log_b n})^d}\right) = a^{\log_b n} = a^{(\log_a n)(\log_b a)} = n^{\log_b a}.$$

3. *The ratio is exactly* 1.
   In this case $O(\log_b n)$ terms of the series are equal to $O(n^d)$.

4. Compare the Kruskal's algorithm and the Prim's algorithm (viz. similarities and differences).

Answer:

	Kruskal	prim
sort all edges?	yes	no
minimum	the lightest edge in the remaining edges	the lightest edge among the cross edges
data structure	disjoint set (connected component)	binary heap (priority queue)
time complexity	$O((\|V\|+\|E\|)\log\|V\|)$	$O((\|V\|+\|E\|)\log\|V\|)$

3. (10 Points) Suppose you are choosing between the following three algorithms:

- **Algorithm *A* solves problems by dividing them into seven subproblems of size *n*/4, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time.**
- **Algorithm *B* solves problems of size *n* by recursively solving one subproblem of size $\sqrt[3]{n}$ and then combining the solution in constant time.**
- **Algorithm *C* solves problems of size *n* by recursively solving a subproblem of size *n*-2 and then combining the solution in $O(c^n)$ time, where $c > 1$ is a constant.**

What are the running times of each of these algorithms (in big-*O* notation), and which would you choose?

Answer:

Algorithm *A*:

$T(n) = 7T(n/4) + O(n^2) = \theta(n^2)$ **by the Master theorem.**

Algorithm *B*:

$$T(n) = T(\sqrt[3]{n}) + c = T(\sqrt[3]{\sqrt[3]{n}}) + c + c = T(\sqrt[3]{\sqrt[3]{\sqrt[3]{n}}}) + 3c$$

$$= k + T(b) \text{ s.t. } b^{\overbrace{3*\ldots*3}^{k}} = n$$

$$\Rightarrow b^{3^k} = n \Rightarrow k = \log_3 \log_b n \Rightarrow k = \theta(\log\log n)$$

Algorithm *C*:

$$T(n) = T(n-2) + O(c^n) = T(n-4) + O(c^{n-2}) + O(c^n)$$
$$= T(n-6) + O(c^{n-4}) + O(c^{n-2}) + O(c^n)$$
$$= T(0) + \sum_{i=1}^{n/2} c^{2i} = T(0) + \theta(c^{2*n/2}) = \theta(c^n)(c^2 > 1)$$

Choose Algorithm *B* due to the lower time complexity.

5. Sort the following functions in the increasing order of asymptotic (big-*O*) complexity and explain why:

$f_1(n) = \log\log n$

$f_2(n) = n^{\sqrt{n}}$

$f_3(n) = \log(n^{100})$

$f_4(n) = 2^{2^{100}}$

$f_5(n) = 2^n$

$f_6(n) = n^{10} \cdot 2^{n/2}$

$f_7(n) = n\sqrt{n}$

$f_8(n) = 2^{2^n}$

$f_9(n) = \sum_{i=1}^{n} i + 1$

$f_{10}(n) = 100n$

Answer:

$f_1(n) = \log\log n$

$f_2(n) = n^{\sqrt{n}} = (2^{\lg n})^{\sqrt{n}} = 2^{\sqrt{n}\cdot\lg n}$

$f_3(n) = \log(n^{100}) = 100\log n = \theta(\log n)$

$f_4(n) = 2^{2^{100}} = \theta(1)$

$f_5(n) = 2^n$

$f_6(n) = n^{10} \cdot 2^{n/2} = 2^{\lg(n^{10})} \cdot 2^{n/2} = 2^{n/2+10\lg n}$

$f_7(n) = n\sqrt{n} = n^{1.5}$

$f_8(n) = 2^{2^n}$

$f_9(n) = \sum_{i=1}^{n} i + 1 = n + (n-1) + \ldots + 1 + 1 = \theta(n^2)$

$f_{10}(n) = 100n = \theta(n)$

so, $f_4(n) < f_1(n) < f_3(n) < f_{10}(n) < f_7(n) < f_9(n) < f_2(n) < f_6(n) < f_5(n) < f_8(n)$

Student Name _____, Student No._____

Score	Evaluator	Section IV: Design and Analysis （50 Points）

1. (10 Points) Give a simple reduction from RUDRATA CYCLE to SAT. (Hint: you may use variables x_{ij} whose intuitive meaning is "vertex *i* is the *j*th vertex of the Hamilton cycle"; you then need to write clauses that express the constraints of the problem.)

Answer:

RUDRATA CYCLE to SAT

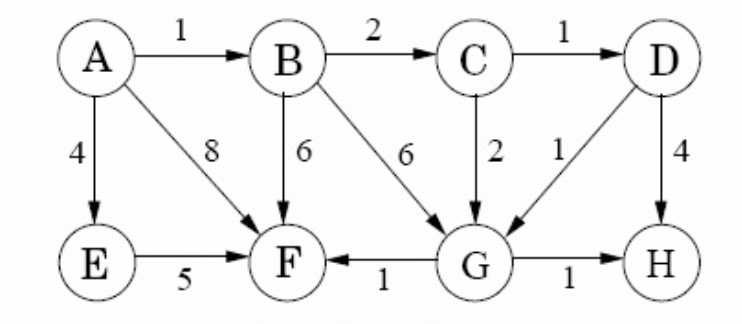
We introduce variables x_{ij} for $1 \leq i, j \leq n$ meaning that the ith vertex is at the jth position in the Rudrata cycle. Each vertex must appear at some position in the cycle. Thus, for every vertex i, we add the clause $x_{i1} \vee x_{i2} \vee \ldots \vee x_{in}$. This adds n clauses with n variables each.

Also, if the ith vertex appears at the jth position, then the vertex at $(j+1)$th position must be a neighbor of i. In other words, if u, v are *not* neighbors, then either u appears at the jth position, or v appears at the $(j+1)$th position, but not both. Thus for every $(u, v) \notin E$ and for all $1 \leq j \leq n$, add the clause $(\bar{x}_{uj} \vee (\bar{x}_{v(j+1)})$. This adds at most $O(n^2) \times n = O(n^3)$ clauses with 2 variables each.

Using the "meanings" of the clauses given above, it is easy to see that every satisfying assignment gives a Rudrata cycle and vice-versa.

(Each clause category: 5 points)

2. (12 Points) Suppose Dijkstra's algorithm is run on the following graph, starting at node *A*.



(1) (4 Points) Draw a table showing the intermediate distance values of all the nodes at each iteration of the algorithm.

(2) (4 Points) Show the final shortest-path tree.

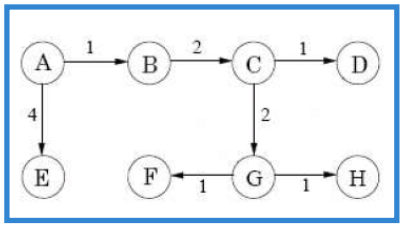
(3) (4 Points) Suppose the above graph becomes undirected. What is the minimal spanning tree, by running Kruscal's algorithm in this undirected graph?

Answer:

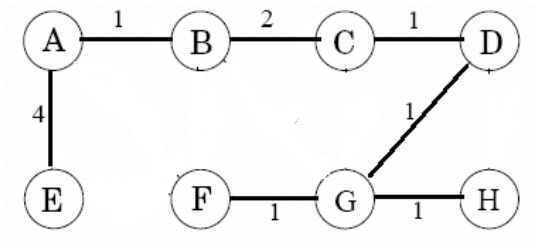
(a)

Node	Iteration							
	0	1	2	3	4	5	6	7
A	0	0	0	0	0	0	0	0
B	∞	1	1	1	1	1	1	1
C	∞	∞	3	3	3	3	3	3
D	∞	∞	∞	4	4	4	4	4
E	∞	4	4	4	4	4	4	4
F	∞	8	7	7	7	7	6	6
G	∞	∞	7	5	5	5	5	5
H	∞	∞	∞	∞	8	8	6	6

(b)



(c) first sort all the edges in an increasing ordering of weights:
A-B, C-D, D-G, G-F, G-H, B-C, C-G, A-E, D-H, E-F, B-F, B-G, A-F
The chosen edges and the final MST are as follows:
A-B, C-D, D-G, G-F, G-H, B-C, A-E



3. (10 Points) Given two strings $x = x_1 x_2 ... x_n$ **and** $y = y_1 y_2 ... y_m$**, we wish to find the length of their longest common substring, that is, the largest *k* for which there are indices *i* and *j* with** $x_i x_{i+1} ... x_{i+k-1} = y_j y_{j+1} ... y_{j+k-1}$ **.**

(1) (6 Points) Give a dynamic programming algorithm to do this in time *O*(*mn*). (Hint: define subproblem *L*(*i*, *j*) to be the length of the longest common substring of *x* and *y* terminating at x_i and y_j)

(2) (4 Points) Suppose that x=ABCBDAB and y=BDCABA. Fill the following table showing all intermediate optimal values $L(i, j)(1 \le i \le 7, 1 \le j \le 6)$ **defined above.**

L(i, j)		**A**	**B**	**C**	**B**	**D**	**A**	**B**
B								
D								
C								
A								
B								
A								

Answer:

(1)

Subproblems: For $1 \leq i \leq n$ and $1 \leq j \leq m$, define subproblem $L(i, j)$ to be the length of the longest common substring of x and y *terminating* at x_i and y_j. The recursion is:

$$L(i,j) = \begin{cases} L(i-1, j-1) + 1 & \text{if equal}(x_i, y_j) = 1 \\ 0 & \text{otherwise} \end{cases}$$

The initialization is, for all $1 \leq i \leq n$ and $1 \leq j \leq m$:

$$L(0,0) = 0$$
$$L(i,0) = 0$$
$$L(0,j) = 0$$

The output of the algorithm is the maximum of $L(i, j)$ over all $1 \leq i \leq n$ and $1 \leq j \leq m$.

Correctness and Running Time: The initialization is clearly correct. Hence, it suffices to prove the correctness of the recursion. The longest common substring terminating at x_i and y_j must include x_i and y_j: hence, it will be 0 if these characters are different and $L(i-1, j-1)+1$ if they are equal. The running time is $O(mn)$ as we have mn subproblems and each takes constant time to evaluate through the recursion.

(2)

X/Y		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	0	1	0	0	1
D	0	0	0	0	0	2	0	0
C	0	0	0	1	0	0	0	0
A	0	1	0	0	0	0	1	0
B	0	0	2	0	1	0	0	2
A	0	1	0	0	0	0	1	0

4. (10 points) A *contiguous* subsequence of a list *S* is a subsequence made up of consecutive elements of S. For instance, if *S* is

5, 15, -30, 10, -5, 40, 10,

then 15, -30, 10 is a contiguous subsequence but 5, 15, 40 is not. Give an algorithm for the following task:

***Input*: A list of numbers, $a_1, a_2, \ldots, a_n$.**

***Output*: The contiguous subsequence of maximum sum (a subsequence of length zero has sum zero).**

For the preceding example, the answer would be 10, -5, 40, 10, with a sum of 55.

Notice that you will get a different score if your algorithm meets a different time complexity requirement. The max score which you can obtain is given for each requirement.

- **Basic requirement: $O(n^2)$ (max score: 6 points)**
- **Medium requirement: $O(n\log n)$ (max score: 8 points)**
- **Advanced requirement: $O(n)$ (max score: 10 points)**

Answer:

1) A brute-force algorithm: $O(n^2)$

We can easily devise a brute-force solution to this problem: just try every possible pair of buy and sell dates in which the buy date precedes the sell date. A period of n days has $\binom{n}{2}$ such pairs of dates. Since $\binom{n}{2}$ is $\Theta(n^2)$, and the best we can hope for is to evaluate each pair of dates in constant time, this approach would take $\Omega(n^2)$ time.

2) A divide-and-conquer algorithm: $O(n\log n)$

```
FIND-MAXIMUM-SUBARRAY(A, low, high)
 1  if high == low
 2      return (low, high, A[low])          // base case: only one element
 3  else mid = ⌊(low + high)/2⌋
 4      (left-low, left-high, left-sum) =
            FIND-MAXIMUM-SUBARRAY(A, low, mid)
 5      (right-low, right-high, right-sum) =
            FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
 6      (cross-low, cross-high, cross-sum) =
            FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
 7      if left-sum ≥ right-sum and left-sum ≥ cross-sum
 8          return (left-low, left-high, left-sum)
 9      elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
10          return (right-low, right-high, right-sum)
11      else return (cross-low, cross-high, cross-sum)
```

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
 1  left-sum = −∞
 2  sum = 0
 3  for i = mid downto low
 4      sum = sum + A[i]
 5      if sum > left-sum
 6          left-sum = sum
 7          max-left = i
 8  right-sum = −∞
 9  sum = 0
10  for j = mid + 1 to high
11      sum = sum + A[j]
12      if sum > right-sum
13          right-sum = sum
14          max-right = j
15  return (max-left, max-right, left-sum + right-sum)
```

3) A dynamic-programming algorithm: $O(n)$

Subproblems: Define an array of subproblems $D(i)$ for $0 \leq i \leq n$. $D(i)$ will be the largest sum of a (possibly empty) contiguous subsequence ending exactly at position i.

Algorithm and Recursion: The algorithm will initialize $D(0) = 0$ and update the $D(i)$'s in ascending order according to the rule:

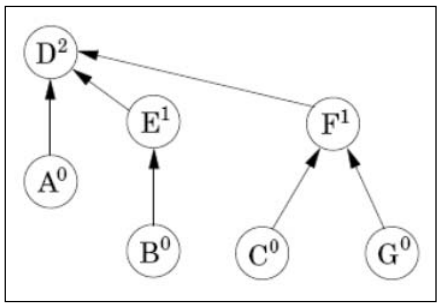
$$D(i) = \max\{0, D(i-1) + a_i\}$$

The largest sum is then given by the maximum element $D(i)^*$ in the array D. The contiguous subsequence of maximum sum will terminate at i^*. Its beginning will be at the first index $j \leq i^*$ such that $D(j-1) = 0$, as this implies that extending the sequence before j will only decrease its sum.

Correctness: The contiguous subsequence of largest sum ending at i will either be empty or contain a_i. In the first case, the value of the sum will be 0. In the second case, it will be the sum of a_i and the best sum we can get ending at $i-1$, i.e. $D(i-1) + a_i$. Because we are looking for the largest sum, $D(i)$ will be the maximum of these two possibilities.

Running Time: The running time for this algorithm is $O(n)$, as we have n subproblems and the solution of each can be computed in constant time. Moreover, the identification of the optimal subsequence only requires a single $O(n)$ time pass through the array D.

5. (10 Points) Prove that if there are n elements overall in disjoint-set trees, there can be at most $n/2^k$ nodes of rank k ($k \geq 0$). (*Hint*: as an example, a disjoint-set tree is shown as follows:)



Answer: Using induction:

k=0: forest of *n* singleton trees with height 0.
k=1: *n*/2 single-child trees with height 1.
assume when rank = *k*, the property holds
how to produce the most nodes at rank *k*+1? Merge equal-height trees as many as possible so that #nodes of rank *k*+1 is at most $(n/2^k)/2$.

6. (10 points) A *vertex cover* of a graph $G = (V, E)$ is a subset of vertices $S \subseteq V$ that includes at least one endpoint of every edge in E. Give a 2-approximation algorithm for the following task. (*Hint*: you need to explain why the approximation ratio of your algorithm is 2.)

Input: An undirected graph $G = (V, E)$.

Output: The size of the smallest vertex cover of G.

Answer:

Approximation Algorithm For Vertex Cover

Here we define algorithm *Approx_Vertex_Cover*, an approximation algorithm for Vertex Cover. Start with an empty set V'. While there are still edges in E, pick an edge (u, v) arbitrarily. Add both u and v into V'. Remove all edges incident on u or v. Repeat until there are no more edges left in E. *Approx_Vertex_Cover* runs in polynomial time.

Claim: *Approx_Vertex_Cover* is a 2-approximation algorithm.

Proof: Let $U \subseteq E$ be the set of all the edges that are picked by *Approx_Vertex_Cover*. The optimal vertex cover must include at least one endpoint of each edge in U (and other edges). Furthermore, no two edges in U share an endpoint. Therefore, $|U|$ is a lower bound for C_{opt}. i.e. $C_{opt} \geq |U|$. The number of vertices in V' returned by *Approx_Vertex_Cover* is $2 \cdot |U|$. Therefore, $C = |V'| = 2 \cdot |U| \leq 2C_{opt}$. Hence $C \leq 2 \cdot C_{opt}$. □