CS570

Analysis of Algorithms

Fall 2004

Midterm

1) Briefly describe the following algorithm techniques. (20 pts)
   a) Greedy method
   b) Dynamic programming
   c) Divide and conquer

What is the commonality between the two approaches of dynamic programming and divide and conquer?

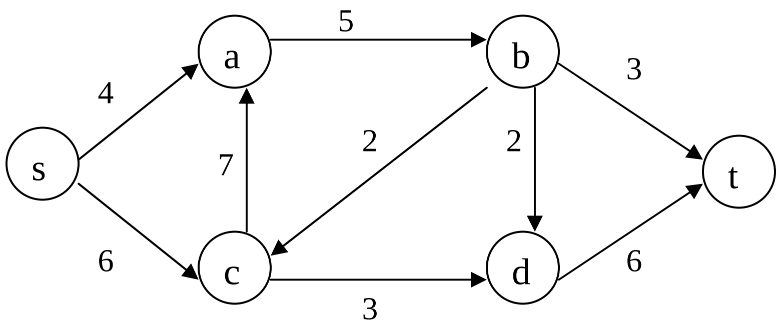
What is the fundamental difference between divide and conquer and dynamic programming?

2) Suppose that a graph G has a minimum spanning tree already computed. How quickly can the MST be updated if a new vertex and incident edges are added to G. (20 pts)

3) Give an example of a directed graph with negative weight edges for which Dijkstra's algorithm produces incorrect answers. Identify the node(s) to which Dijstra's algorithm incorrectly finds the shortest path. Explain why. (20 pts)

4) Consider the following scenario, you are given all the daily closing prices of shares of Microsoft over a period of n days. You are given a chance to buy 100 shares of this stock at any day i, $1 \le i \le n$ and sell at any day j, $i \le j \le n$. Of course your objective is to maximize your profit in this transaction. There is a simplistic $O(n^2)$ algorithm that tries all possible pairs of buy/sell days to find the optimal solution. Develop an algorithm that accomplishes this in O(n) time. (20 pts)

5) Consider the following flow network. (20 pts)



a) What is the maximum value of an (s,t) flow in the above flow network?
b) Identify a minimum s-t cut in this flow network, what is the capacity of this cut?
c) Is the minimum s-t cut unique, i.e. are there other s-t cuts in this graph with same capacity?

d) Give an example of a network with real numbers for edge capacities (as opposed to integer numbers) where the Ford-Fulkerson algorithm may not terminate. Explain the reason why the algorithm may not terminate

CS570

Analysis of Algorithms

Spring 2005

Midterm Exam

Name: ______________________

Student ID: ___________________

1) 12 pts
Considering an undirected graph G=(V, E), are the following statements true or false? Provide a brief explanation. The space provided should suffice.

A- Suppose all edge weights are different. Then the shortest edge must be in the minimum spanning tree.

B- Suppose all edge weights are different. Then the longest edge cannot be in the minimum spanning tree.

C- Suppose all edge weights are different. Then the minimum spanning tree is unique.

D- Suppose all edge weights are different. Then the shortest path from A to B is unique.

2) 9 pts
Considering amortized and actual cost analysis of heap operations {insert, extract-min, union, decrease-key, delete-key} on heap data structures we have studied. are the following statements true or false? Provide a brief explanation. The space provided should suffice.

A- **Actual** cost of any operation in a **Fibonacci** heap data structure takes at most O(log n) time.

B- **Actual** cost of any operation in a **binomial** heap data structure takes at most O(log n) time.

C- **Actual** cost of any operation in a **binary** heap data structure takes at most O(log n) time.

3) 20 pts

A- Give an example of an undirected graph **G** with positive edge weights and a starting vertex **s** in **G** for which Dijktra's and Prim's algorithms visit nodes in different orders.

B- Show the order in which nodes in **G** (the graph you presented in part A) are visited in both algorithms.

C- Show the minimum spanning tree resulting from the execution of Prim's algorithm on your graph.

4) 20 pts
Consider the following Change Problem. The input to this problem is an integer *L*. The output should be the minimum cardinality collection of coins required to make *L* shillings of change (that is, you want to use as few coins as possible). The coins are worth 1, 5, 10, 20, 25, 50 shillings. Assume that you have an unlimited number of coins of each type.

Does the following greedy algorithm correctly solve the problem?
Take as many coins as possible from the highest denominations.
If it does, prove the accuracy of the solution. If it does not, give a counter example.

5) 20 pts
For the Change problem defined in the previous question, what other solution technique (other than greedy) seems appropriate?

A- Present your solution using an alternate technique.

B- Analyze the running time of the algorithm presented above.

6) 20 pts

Using a straightforward approach, finding both the minimum and the maximum elements of a set of **n** numbers can take **2*(n-1)** comparisons, i.e. finding the minimum element takes **n-1** comparisons and then to find the maximum element takes another **n-1** comparisons. We seek to reduce the number of comparisons using a divide and conquer approach.

A- Can the number of comparisons be improved by any method to be lower than **O(n)**? If yes, demonstrate how, if no prove why not.

B- Can we improve on the constant ***a*** involved in the ***a*n+*b*** number of comparisons? In other words can we reduce **2*n-2** to ***a*n+*b*** where ***a*** is less than **2**.

CS570
Analysis of Algorithms
Summer 2005
Midterm Exam

Name: ________________________
Student ID: ____________________

1) 20 pts
Binomial min-heap H1 contains elements {12, 7, 25, 15, 28, 41, 33}, and binomial min-heap H2 contains elements {18, 3, 37, 6, 8, 30, 45, 55, 32, 23, 24, 22, 29, 48, 50, 31, 10, 17, 44}.
a) Show heaps H1 and H2 (Note: there could be more than one possible correct construction of the heap)

b) Merge H1 and H2 into a new Binomial min-heap H. Show all you work.

2) 20 pts

Suppose that you are playing a very simple arcade game. You start at the left end of the screen (position 0), and are supposed to make it to the right end (position k). In each step, you can jump between 1- 9 positions to the right (but never to the left). The problem is that some of those positions contain harmful things, like fires, monsters, rotating knives, CS 570 midterms, etc. When you touch such a position, you lose some of your “life energy".

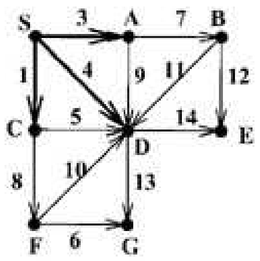
More formally, when you touch position i (with $0 \le i \le k$), you lose $x_i \ge 0$ points of energy (notice that you don't lose any energy if you jump over a position). Now you are supposed to find a sequence of jumps that gets you from 0 to k while losing the smallest total amount of energy.

Give an algorithm to compute, in polynomial time, the smallest total amount of energy you can lose going from 0 to k. Analyze the running time of your algorithm.

3) 20 pts

We are running one of these three algorithms on the graph below, where the algorithm has already "processed" the bold-face edges (SA, SD, and SC). (Ignore the directions on the edges for Prim's and Kruskal's algorithms.)

- Prim's for the minimum spanning tree, starting from S.
- Kruskal's for the minimum spanning tree.
- Dijkstra's for shortest path from S.



a) Which two edges would be added next in Prim's algorithm? Be sure to indicate the order in which they are added.

b) Which two edges would be added next in Kruskal's algorithm? Be sure to indicate the order in which they are added.

c) At this point in the running of Dijkstra's algorithm, S has been taken off the top of the heap and marked. Which four vertices would be marked next in Dijkstra's algorithm, i.e. deleted from the top of the heap? Be sure to indicate the order in which they are deleted. Which final edges would Dijkstra's algorithm choose as part of the shortest path to these vertices (i.e. which edge connects to this vertex as part of the shortest path from S)?

4) 20 pts
Consider the Change Problem in Binaryland The input to this problem is an integer *L*. The output should be the minimum cardinality collection of coins required to make *L* nibbles of change (that is, you want to use as few coins as possible).

In Binaryland the coins are worth $1, 2, 2^2, 2^3, \ldots, 2^{1000}$ nibbles. Assume that you have an unlimited number of coins of each type. Prove or disprove that the greedy algorithm (that takes as many coins of the highest value as possible) solves the change problem in Binaryland.

5) 20 pts

You have successfully implemented an algorithm using divide and conquer to find the kth smallest term in an unsorted array of n elements. Your algorithm has a divide step that takes O(n) time during which it divides the problem into 4 equal pieces. It then forms 3 sub-problems that it solves recursively. The sub-problems are then combined in O(lg n).

Determine if this algorithm has a better complexity than the straight-forward method of first sorting the array and then finding the kth smallest term in the sorted array. Show all your work.

Name(last, first): ___________________

CS570
Analysis of Algorithms
Fall 2005
Midterm Exam

Name: ________________________
Student ID: ____________________

	Maximum	Received
Problem 1	14	
Problem 2	2	
Problem 3	5	
Problem 4	10	
Problem 5	10	
Problem 6	15	
Problem 7	20	
Problem 8	15	
Problem 9	9	

Name(last, first): ___________________

1. 14 pts
   Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification.

   [TRUE/FALSE]
   A dynamic programming algorithm tames the complexity by making sure that no subproblem is solved more than once.

   [TRUE/FALSE]
   The memoization approach in dynamic programming has the disadvantage that sometimes one may solve subproblems that are not really needed.

   [TRUE/FALSE]
   A greedy algorithm finds an optimal solution by making a sequence of choices and at each decision point in the algorithm, the choice that seems best at the moment is chosen.

   [TRUE/FALSE]
   If a problem can be solved correctly using the greedy strategy, there will only be one greedy choice (such as "choose the object with highest value to weight ratio") for that problem that leads to the optimal solution.

   [TRUE/FALSE]
   Whereas there could be many optimal solutions to a combinatorial optimization problem, the value associated with them will be unique.

   [TRUE/FALSE]
   Let G be a directed weighted graph, and let u, v be two vertices. Then a shortest path from u to v remains a shortest path when 1 is added to every edge weight.

   [TRUE/FALSE]
   Given a connected, undirected graph G with distinct edge weights, then the edge e with the <u>second</u> smallest weight is included in the minimum spanning tree.

2. 2 pts
   In our first lecture this semester, which of the following techniques ware used to solve the Stable Matching problem? Circle all that may apply.
   A- Greedy
   B- Dynamic Programming
   C- Divide and Conquer

Name(last, first): ___________________

3. 5 pts
   A divide and conquer algorithm is based on the following general approach:

   - Divide the problem into 4 subproblems whose size is one third the size of the problem at hand. This is done in O(lg n)
   - Solve the subproblems recursively
   - Merge the solution to subproblems in linear time with respect to the problem size at hand

   What is the complexity of this algorithm?

4. 10 pts
   Indicate for each pair of expressions (A,B) in the table below, whether A is **O**, **Ω**, or **ϴ** of B. Assume that k and c are positive constants.

A	B	**O**	**Ω**	**ϴ**
$(\lg n)^k$	Cn			
2^n	$2^{(n+1)}$			
$2^n n^k$	$2^n n^{2k}$			

Name(last, first): ___________________

5. 10 pts
   The array A below holds a max-heap. What will be the order of elements in array A after a new entry with value 19 is inserted into this heap? Show all your work.
   A = {16, 14, 10, 8, 7, 9, 3, 2, 4, 1}

6. 15 pts
   Consider a max-heap H and a given number X. We want to find whether X is larger than the k-th largest element in the list. Design an O(k) time algorithm to do this.
   Note: you do not need to prove your algorithm but you need to prove the complexity.

Name(last, first): ___________________

7. 20 pts total
   Consider the following arcade game where the player starts at the lower left corner of an n by n grid to reach the top right corner. At each step the player can either jump to the right (x –axis) or jump up (y-axis). Depending on his/her energy level the player can jump either 1 or 2 squares. If his/her energy level is higher than E he/she can jump 2 squares, otherwise only one square. When landing on square (i,j) the player may gain or lose energy equal to $|X_{ij}|$. Positive X_{ij} indicates energy gain. In addition, every time the player jumps 2 squares he/she loses E/2 units of energy.

   a) Present a polynomial time solution to find the highest energy level the player can end the game with. (15 pts)

b) Describe how you could build the path that leads to the optimal solution. (5 pts)

8. 15 pts
   T is a spanning tree on an undirected graph G=(V,E). Edge costs in G are NOT guaranteed to be unique. Prove or disprove the following:
   If every edge in T belongs to SOME minimum cost spanning tree in G, then T is itself a minimum cost spanning tree.

9. 9 pts
   We have shown in class that the Integer Knapsack problem can be solved in O(nW) where n is the number of items and W is the capacity of the knapsack. The Fractional Knapsack problem is less restrictive in that it allows fractions of items to be placed in the knapsack. Solve the Fractional Knapsack problem (have n objects, weight of object i = W_i, value of object i = V_i , weight capacity of knapsack=W, Objective: Fill up knapsack with objects to maximize total value of objects in knapsack)

CS570 MIDTERM SOLUTION

Q 1:
1.True
2.False
3.True
4.False
5.True
6.False
7.True
Grading Policy : 2 points and all or none for each question.

Q 2: The answer is A

Grading Policy : Any other solution is wrong and get zero

Q 3:

$T(n) = 4T(\frac{n}{3}) + \lg n + cn$ by master theorem, $T(n)$ is $O(n^{\log_3 4})$
Grading Policy : 2 points for the correct recursive relation. 3 points for the correct result $O(n^{\log_3 4})$. If the student provide the correct result but doesn't give the recursive relation, get 4 points.

Q 4: Click ✓on $(1,1),(2,1),(2,2),(2,3),(3,1)$ and write $\times$ on others . Here (i,j) means the $i-th$ row and $j-th$ column in the blank form.
Grading Policy : 1 points for each entry in the blank form

Q 5: The process of a new entry with value 19 inserted is as follows:

(1) A={16,14,10,8,7,9,3,2,4,1,19}
(2) A={16,14,10,8,19,9,3,2,4,1,7}
(3) A={16,19,10,8,14,9,3,2,4,1,7}
(4) A={19,16,10,8,14,9,3,2,4,1,7}

Grading Policy : It's OK if represent A by a graph. 2.5 points for each step.
Q 6:

Algorithm: The max head is a tree and we start from the root doing the BFS(or DFS) search. If the incident node is bigger than x, we add

Date: Oct 17, 2005.

it into the queue. By doing this if we already find k nodes then x is smaller than at least k nodes in the heap, otherwise, x is bigger than $k-th$ biggest node in the heap.

running time: During the BFS(or DFS), we traverse at most k nodes. Thus the running time is $O(k)$

Grading Policy : 9 points for the algorithm and 6 points for the correct running time. A typical mistake is that many students assumed that the max heap was well sorted and just pick the k-th number in the heap. In this case they get 3 points at most

Q 7:

a) The recursive relation is as follows:

$$OPT(m,n) = max \begin{cases} OPT(m,n-1) + X_{mn} \\ OPT(m,n-2) + X_{mn} - E/2, & \text{if OPT}(m,n-2) > \frac{E}{2} \\ OPT(m-1,n) + X_{mn} \\ OPT(m-2,n) + X_{mn} - E/2, & \text{if } OPT(m-2,n) > \frac{E}{2} \end{cases}$$

And the boundary condition is $OPT(1,1) = E_0$. Clearly to find the value of $OPT(n,n)$, we need to fill an $n \times n$ matrix. It takes constant time to do the addition and compare the 4 numbers in the recursive relation. Hence the running time of our algorithm is $O(n^2)$

b)After the construction of the $n \times n$ table in a), we start right at the upper right corner (n,n). We compare the value with the 4 previous value in the recursive relation mentioned in a). If we get $OPT(n,n)$ from the previous position (i,j) in the recursive relation, store the position (i,j) in the path and repeat the process at (i,j) until we get to $(1,1)$

Grading Policy : For part a), 10 points for the correct recursive relation and 5 points for the correct running time. If the student didn't solve this question by dynamic programming, he gets at most 4 points as the partial credit.

For part b), make sure they know the idea behind the problem.

Q 8: False. The counter example is as follows: Consider the graph $(a,b),(b,c),(a,c)$ where $w(a,b) = 4, w(b,c) = 4, w(c,a) = 2$. Clearly (a,b) is in the MST $((a,b),(c,a))$ and (b,c) is in the MST $((b,c),(c,a))$ but $((a,b),(b,c))$ is not MST of the graph.

Grading Policy : If the answer is "True", at most get 5 points for the whole question. If the answer is "False" but didn't give a counter example, get 9 points. If the answer is "false" and give a correct counter example, but didn't point out the spanning tree in which every edge is in some MST but the spanning tree is not MST, get 12 points.

Q 9: First we calculate $c_i = V_i/W_i$ for $i = 1, 2, ..., n$, then we sort the objects by decreasing c_i. After that we follows the greedy strategy of always taking as much as possible of the objects remaining which has highest c_i until the knapsack is full. Clearly calculating c_i takes $O(n)$ and sorting c_i takes $O(n \log g)$. Hence the running time of our algorithm is $O(n) + O(n \log n)$, that is, $O(n \log n)$. We prove our algorithm by contradiction. If $S = \{O_1, ..., O_m\}$ is the objects the optimal solution with weight $w_1, ..., w_m$. The total value in the knapsack is $\sum_{i=1}^{m} (w_i/W_i)V_i$.Assume there is i, j(without loss of generality assume that $i < j$) such that $V_i/W_i > V_j/W_j$ and there is V_i with weight w'_i left outside the knapsack. Then replacing $min\{w'_i, w_j\}$ weight of O_j with $min\{w'_i, w_j\}$ weight of O_i we get a solution with total value $\sum_{k=1}^{m} w_k/W_kV_k + min\{w'_i, w_j\}/W_iV_i - min\{w'_i, w_j\}/W_jV_j$. $w'_i > 0, w_j > 0$ and thus $min\{w'_i, w_j\} > 0$. Note that $V_i/W_i > V_j/W_j$, hence $\sum_{k=1}^{m}(w_k/W_k)V_k + (min\{w'_i, w_j\}/W_i)V_i - (min\{w'_i, w_j\}/W_j)V_j > \sum_{k=1}^{m}(w_k/W_k)V_k$. we get a better solution than the optimal solution. Contradiction!

Grading Policy : 5 points for the algorithm, 2 points for the proof and 2 points for the running time.

Dynamic Programming will not work here since the fractions of items are allowed to be placed in the knapsack. If the student tried solving this problem by dynamic programming and both the recursive relation and time complexity are correct, they get 4 points at most.

CS570
Analysis of Algorithms
Summer 2006
Exam 1

Name: ________________________
Student ID: ____________________

	Maximum	Received
Problem 1	20	
Problem 2	10	
Problem 3	10	
Problem 4	10	
Problem 5	10	
Problem 6	20	
Problem 7	20	

1) 20 pts
Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification.

[TRUE/FALSE]
The running time of an algorithm is θ(g(n)) if and only if its best-case running time is Ω(g(n)) and its average-case running time is O(g(n)).

[TRUE/FALSE]
A dynamic programming algorithm tames the complexity by making sure that no subproblem is solved more than once.

[TRUE/FALSE]
The memoization approach in dynamic programming has the disadvantage that sometimes one may solve subproblems that are not really needed.

[TRUE/FALSE]
A greedy algorithm finds an optimal solution by making a sequence of choices and at each decision point in the algorithm, the choice that seems best at the moment is chosen.

[TRUE/FALSE]
If a problem can be solved correctly using the greedy strategy, there will only be one greedy choice (such as "choose the object with highest value to weight ratio") for that problem that leads to the optimal solution.

[TRUE/FALSE]
Whereas there could be many optimal solutions to a combinatorial optimization problem, the value associated with them will be unique.

[TRUE/FALSE]
Consider an undirected graph G=(V, E). Suppose all edge weights are different. Then the longest edge cannot be in the minimum spanning tree.

[TRUE/FALSE]
Consider an undirected graph G=(V, E). Suppose all edge weights are different. Then the shortest edge must be in the minimum spanning tree.

[TRUE/FALSE]
Consider an undirected graph G=(V, E). Suppose all edge weights are different. Then the shortest path from A to B is unique.

[TRUE/FALSE]
Bellman Ford's algorithms of finding the shorting s-t path is more suitable for parallel processing than Dijkstra's.

2) 10 pts
Indicate for each pair of expressions (A,B) in the table below, whether A is **O**, **Ω**, or **Θ** of B. Assume that k and c are positive constants. You can mark each box with Y (yes) and N (no).

A	B	**O**	**Ω**	**Θ**
$(\lg n)^k$	Cn			
2^n	$2^{(n+1)}$			
$2^n n^k$	$2^n n^{2k}$			

3) 10 pts
Is an array that is in sorted order a min-heap? Explain your answer

4) 10 pts

Given the following nested loop

```
X = 0
For I=1 to n
          For J= 1 to O(n)
                 X = X +1
          Endfor
Endfor
```

a- Can we conclude that at loop termination $X=O(n^2)$? If yes, explain why. If no, give counter example.

b- Can we conclude that at loop termination $O(n^2)$ is a tight upper bound on X? If yes, explain why. If no, give counter example.

5) 10 pts
Give an input instance where Dijsktra's algorithm will not solve the single source shortest path problem but the Bellman-Ford algorithm will.

6) 20 pts
Design an efficeint algorithm to find a spanning tree for a connected weighted undirected graph G=(V,E) such that the weight of the maximum-weight edge in the spanning tree is minimized. Prove its correctness and analyze its complexity.

7) 20 pts
The Subset Addition problem has inputs that include a target value M and weights W[1…n] for n objects. The problem asks whether there exists a subset S of objects with $S \subseteq \{1, 2, \dots, n\}$, such that the sum over $j \in S$ of the weights W[j] is equal to exactly M. The output should be either true (if the answer is yes) or false (if it is no). Provide an O(nM)-time algorithm that solves this Subset Addition problem.

CS570
Analysis of Algorithms
Fall 2006

Exam 1

Name: ________________________

Student ID: ____________________

	Maximum	Received
Problem 1	20	
Problem 2	10	
Problem 3	10	
Problem 4	10	
Problem 5	10	
Problem 6	20	
Problem 7	20	

Note: The exam is closed book closed notes.

1) 20 pts
Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification.

[TRUE/FALSE] True (By definition it is true)
If T(n) is both **O**(f(n)) and **Ω**(f(n)), then T(n) is **Θ**(f(n)) .

[TRUE/FALSE] True
For a graph G and a node v in that graph, the DFS and BFS trees of G rooted at v always contain the same number of edges

[TRUE/FALSE] False (Counter example: Fibonacci Heap)
Complexity of the "Decrease_Key" operation is always O(lgn) for a priority queue.

[TRUE/FALSE] True (See the solution of HW4)
For a graph with distinct edge weights there is a unique MST.

[TRUE/FALSE] True (yes and store all the possible solution in a table)
Dynamic programming considers all the possible solutions.

[TRUE/FALSE] False(The graph a-b-c-d-a with three edges are 1 and the one left is 4)
Consider an undirected graph G=(V, E) and a shortest path P from s to t in G. Suppose we add one 1 to the cost of each edge in G. P will still remain as a shortest path from s to t.

[TRUE/FALSE] True
Consider an undirected graph G=(V, E) and its minimum spanning tree T. Suppose we add one 1 to the cost of each edge in G. T will still remain as an MST.

[TRUE/FALSE] False (Counter example: Shortest Path in a Graph)
Problems solved using dynamic programming cannot be solved thru greedy algorithms.

[TRUE/FALSE] False (union-Find data structure is for Kruskal's algorithm, it is not for the reverse delete. Check the textbook for more details)
The union-Find data structure can be used for an efficient implementation of the reverse delete algorithm to find an MST.

[TRUE/FALSE] False (Prim algorithm Vs Kruskal algorithm, return can be different)
While there are different algorithms to find a minimum spanning tree of undirected connected weighted graph G, all of these algorithms produce the same result for a given G.

2) 10 pts

Indicate for each pair of expressions (A,B) in the table below, whether A is **O**, **Ω**, or **Ɵ** of B. Assume that k and c are positive constants. You can mark each box with Y (yes) and N (no).

A	B	**O**	**Ω**	**Ɵ**
$n^3 + n^2 + n + c$	n^3	Yes	Yes	Yes
2^n	$2^{(n+k)}$	Yes	Yes	Yes
n^2	$n \cdot 2^{\log(n)}$	No	Yes	No

3) 10 pts

a- What is the minimum and maximum numbers of elements in a heap of height h?

The minimum is 2^(n-1)
The maximum number is 2^n-1

b- What is the number of leaves in a heap of size *n*?
The smallest integer that no less than n/2.
Or
When n is even, the number of leave is n/2;
When n is odd; the number of leaver is (n+1)/2

c- Is the sequence < 23, 7, 14, 6, 13, 10, 1, 5, 17, 12 > a max-heap? If not, show how to heapify the sequence.
Answer: No. The sequence in heapify is
< 23, 7, 14, 17, 13, 10, 1, 5, 6, 12 >, < 23, 17, 14, 7, 13, 10, 1, 5, 6, 12 >

d- Where in a max-heap might the smallest element reside, assuming that all elements are distinct.
The smallest element might reside in any leaves

Grading policy: 2 points for a); 2 points for b); 3 points for c); 3 points for d)

4) 10 pts
Prove or disprove the following:
The shortest path between any two nodes in the minimum spanning tree T = (V, E') of connected weighted undirected graph G = (V,E) is a shortest path between the same two nodes in G. Assume the weights of all edges in G are unique and larger than zero.
False. Consider the graph A-B-C-D-A with weight 1,2,3,4

Grading policy:
-Any one says that the statement is true and try to prove it (of course with wrong proof) may get partial credit up to 3 marks.
- Any one says it is false without correct disproof (counter example) may get partial credit up to 6 marks.
- In Question 5, any one says it is false and give a counter example that has one or more nodes in both G1 and G2 may get partial credit up to 6 marks. Because that is a mistake, nodes in G1 can not exist in G2 and vice versa.
- Any one says it is false and give correct disprove (counter example) get 10 out of 10.

5) 10 pts
Suppose that you divided a graph G = (V,E) into two sub graphs G1 = (V1,E1) and G2 = (V2, E2). And, we can find M1 which is a MST of G1 and M2 which is MST of G2. Then, M1 U M2 U {minimum weight edge among those connecting two graph G1 and G2} always gives MST of G. Prove it or disprove it.
Solution: False.
Counter example: Consider a graph G composed of 4 nodes: A,B,C,D with edge w(A,B)=1,w(B,C)=1, w(B,D)=1, w(C,D)=2. Let V1={A,B} with edge AB, V2={C,D} with edge CD. Then the weight of M1 U M2 U is 4 while the weight of the MST of G is 3.

Grading policy:
-Any one says that the statement is true and try to prove it (of course with wrong proof) may get partial credit up to 3 marks.
- Any one says it is false without correct disproof (counter example) may get partial credit up to 6 marks.
- In Question 5, any one says it is false and give a counter example that has one or more nodes in both G1 and G2 may get partial credit up to 6 marks. Because that is a mistake, nodes in G1 can not exist in G2 and vice versa.
- Any one says it is false and give correct disprove (counter example) get 10 out of 10.

6) 20 pts
There are n workers in the factory with heights of $p_1, p_2, \ldots, p_n$, and n working-clothes with height sizes of $s_1, s_2, \ldots, s_n$. The problem is to find best matching strategy such that we minimize the following average differences.

$$\frac{1}{n}\Sigma|p_i - s_i|$$

Present an algorithm to solve this problem along with its proof of correctness.
Solution: Sort the height of workers in increasing order $p_1 \le p_2 \le \ldots \le p_n$
Sort the height size of clothes in increasing order $s_1 \le s_2 \le \ldots \le s_n$
Match h_i with s_i is the best matching strategy such that we minimize the following average differences.

$$\frac{1}{n}\Sigma|p_i - s_i|$$

The algorithm is correct for the problem of minimizing the average difference between the heights of workers and their clothes. The proof is by contradiction. Assume the people and clothes are numbered in increasing order by height. If the greedy algorithm is not optimal, then there is some input p1, . . . , pn, s1, . . . , sn for which it does not produce an optimal solution. Let the optimal solution be T = {(p1, s_(1)), . . . , (pn, s_(n))}, and let the output of the greedy algorithm be G = {(p1, s1), . . . , (pn, sn)}. Beginning with p1, compare T and G. Let pi be the first person who is assigned different cloth in G than in T. Let sj be the pair of cloth assigned to pi in T. Create solution T' by switching the cloth assignments of pi and pj . By the definition of the greedy algorithm, si is equal or greater than sj . The total cost of T' is given by

$$Cost(T') = Cost(T) - \frac{1}{n}(|p_i - s_j| + |p_j - s_i| - |p_i - s_i| - |p_j - s_j|)$$

There are six cases to be considered. For each case, one needs to show that $(|p_i - s_j| + |p_j - s_i| - |p_i - s_i| - |p_j - s_j|) \geq 0$.

Case 1: $p_i \leq p_j \leq s_i \leq s_j$.

$$\begin{aligned} |p_i - s_j| + |p_j - s_i| - |p_i - s_i| - |p_j - s_j| &= \\ (s_j - p_i) + (s_i - p_j) - (s_i - p_i) - (s_j - p_j) &= 0 \end{aligned}$$

Case 2: $p_i \leq s_i \leq p_j \leq s_j$.

$$\begin{aligned} |p_i - s_j| + |p_j - s_i| - |p_i - s_i| - |p_j - s_j| &= \\ (s_j - p_i) + (p_j - s_i) - (s_i - p_i) - (s_j - p_j) &= \\ 2(p_j - s_i) &\geq 0 \end{aligned}$$

Case 3: $p_i \leq s_i \leq s_j \leq p_j$.

$$\begin{aligned} |p_i - s_j| + |p_j - s_i| - |p_i - s_i| - |p_j - s_j| &= \\ (s_j - p_i) + (p_j - s_i) - (s_i - p_i) - (p_j - s_j) &= \\ 2(s_j - s_i) &\geq 0 \end{aligned}$$

Case 4: $s_i \leq s_j \leq p_i \leq p_j$.

$$\begin{aligned} |p_i - s_j| + |p_j - s_i| - |p_i - s_i| - |p_j - s_j| &= \\ (p_i - s_j) + (p_j - s_i) - (p_i - s_i) - (p_j - s_j) &= 0 \end{aligned}$$

Case 5: $s_i \leq p_i \leq s_j \leq p_j$.

$$\begin{aligned} |p_i - s_j| + |p_j - s_i| - |p_i - s_i| - |p_j - s_j| &= \\ (s_j - p_i) + (p_j - s_i) - (p_i - s_i) - (p_j - s_j) &= \\ 2(s_j - p_i) &\geq 0 \end{aligned}$$

Case 6: $s_i \leq p_i \leq p_j \leq s_j$.

$$\begin{aligned} |p_i - s_j| + |p_j - s_i| - |p_i - s_i| - |p_j - s_j| &= \\ (s_j - p_i) + (p_j - s_i) - (p_i - s_i) - (s_j - p_j) &= \\ 2(p_j - p_i) &\geq 0 \end{aligned}$$

Running time: Sorting takes O(nlog n) and hence the running time is O(nlog n)

Grading policy: Correct description of algorithm 10 pts
Based on the above correct algorithm, present correct complexity 4pts
Correct proof (all six cases) 6pts

7) 20 pts
Given an unlimited supply of coins of denominations $x_1, x_2, \ldots, x_n$, we wish to make change for a value *v*, that is, we wish to find a set of coins whose total value is *v*. This might not be possible: for example, if the denominations are 5 and 10 then we can make change for 15 but not for 12. Give an *O(nv)* algorithm to determine if it is possible to make change for *v* using coins of denominations $x_1, x_2, \ldots, x_n$.

Solution: We solve this question by dynamic programming. Denote OPT(i) as the possibility of paying value i using coins of denominations $x_1, x_2, \ldots, x_n$. Then OPT(i) is true if and only if we can paying $i\text{-}x_{1,}$ or $i\text{-}x_2$, …or $i\text{-}x_n$ using coins of denominations $x_1, x_2, \ldots, x_n$. In other words,
$OPT(i)= OPT(i- x_1)\vee\ OPT(i- x_2)\vee \ldots\ OPT(i- x_n)$
Following the recursive relation with initial value $OPT(x_1)=\ldots =OPT(x_n)$=true we compute the value of OPT(v). If OPT(v) is true, then we can change for *v* using coins of denominations $x_1, x_2, \ldots, x_n$, otherwise we can not.

Clearly to compute OPT(v) we need an array of size v, and each time when we compute OPT(i), we do n union operations. Hence the running time is O(nv)

The following pseudo-code would help you understand the solution:
Data structures: A[v,n]: 2-dimensional array with entries T or F;
OPT[1..n]: 1-dimensional array with entries T or F.

```
 for (j=1 to v)
 OPT(j) = F;
 for (j=1 to v)
 for (i=1 to n)
{
    if (j < xi) then A[j,i] = F;
    if (j == xi) then A[j,i] = T;
    if (j > xi) then A[j,i] = OPT(xi) AND OPT(j-xi);
    if (A[j,i] == T) then OPT(j) = T;
}
return OPT(v)
```

CS570
Analysis of Algorithms
Spring 2007
Exam 1

Name: ______________________
Student ID: __________________

	Maximum	Received
Problem 1	14	
Problem 2	6	
Problem 3	15	
Problem 4	20	
Problem 5	15	
Problem 6	20	
Problem 7	10	

Note: The exam is closed book closed notes.

1) 14 pts
Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification.

[TRUE/FALSE]False
Function $f(n)= 5n^3 2^n + 6n^2 3^n$ is $O(n^3 2^n)$.

[TRUE/FALSE] True
$n\log^3 n$ is NOT $O(n\log n)$

[TRUE/FALSE] True
In an undirected graph, the shortest path between two nodes lies on some minimum spanning tree.

[TRUE/FALSE] False
Max base heap can be converted into Min based heap by reversing the order of the elements in the heap array.

[TRUE/FALSE] False
Kruskal's algorithm for finding a MST of a weighted, undirected graph is a form of a dynamic programming technique.

[TRUE/FALSE] True
In the case of applying Dijkstra's algorithm for dense graph, Fibbonacci implementation is the best one among Binary, Binomial, and Fibbonacci.

[TRUE/FALSE] False
If all of the weights from a connected and weighted graph are distinct, then distinct spanning trees of the graph have distinct weights.

2) 6pts

What is the worst-case complexity of the each of the following code fragments?

a)
```
for (i = 0; i < 2N; i++) {
      sequence of statements
     }
for (j = 0; j < 3M; j++) {
    sequence of statements
     }
```
O(M+N)

b)
```
for (i = 0; i < N; i++) {
          for (j = 0; j < 2N^2; j++) {
              sequence of statements
          }
      }
      for (k = 0; k < 3M; k++) {
          sequence of statements
      }
```
$O(N^3+M)$

c)
```
for (i = 0; i < N^3; i++) {
          for (j = i; j < 2lg(M); j++) {
              sequence of statements
          }
      }
      for (k = 0; k < M; k++) {
              sequence of statements
            }
```

$O(N^3 \log M+M)$

3) 15 pts
The maximum spanning tree problem is to find a spanning tree of a graph whose edge weights have the largest sum possible. Give an algorithm to find a maximum spanning tree and give a proof that the algorithm is correct.
Solution: We can reduce finding the maximum spanning tree problem into finding minimum spanning tree problem. The reduction is as follows: Let w(e) denote the weight of edge e in the graph and let M be the maximum weight of all the edges in G. Now consider re-weighting each edge e in G as M-w(e). Now consider two spanning tree of G, T and T' and note that they both contain exactly the same number of edges, i.e., n-1 edges, where G has n nodes. The key point is that $\sum_{e\in T} w(e) \geq \sum_{e'\in T'} w(e')$ if and only if

$$\sum_{e\in T}[M - w(e)] = (n-1)M - \sum_{e\in T} w(e) \leq (n-1)M - \sum_{e'\in T'} w(e') = \sum_{e'\in T'}[M - w(e')]$$

Hence, if we solve the minimum spanning tree problem using the new edge weights, the optimal tree found will be the maximum spanning tree using the original weights. Therefore we can reduce the maximum spanning tree problem into the minimum spanning tree problem. Hence we can use either Kruskal's algorithm or Prim's algorithm after the reduction and the running time is

O(mlog n)

4) 20 pts
Suppose you are given two set *A* and *B*, each containing *n* positive integers. You can choose to reorder each set however you like. After reordering, let a_i be the *i*-th element of set *A*, and let b_i be the *i*th element of set *B*. You then receive a payoff of $\sum^{n}_{i=1} b_i \log a_i$. Give an algorithm that will maximize your payoff. Prove that your algorithm maximizes the payoff, and state its running time.

Algorithm: Sort A and B in increasing order and the payoff is maximized

Proof: Suppose $\{a_1,\ldots,a_n\}$ and $\{b_1,\ldots b_n\}$ is an optimal solution but $(a_i>a_j)$ and $(b_i<b_j)$. Then switch a_i and a_j we can get a better solution. The reason is as follows:

$$a_i > a_j, b_i < b_j \Leftrightarrow a_i^{b_j-b_i} > a_j^{b_j-b_i} \Leftrightarrow a_i^{b_j} a_j^{b_i} > a_j^{b_j} a_i^{b_i} \Leftrightarrow \log(a_i^{b_j} a_j^{b_i}) > \log(a_j^{b_j} a_i^{b_i})$$
$$\Leftrightarrow b_j \log a_i + b_i \log a_j > b_j \log a_j + b_i \log a_i$$

which contradicts that $\{a_1,\ldots,a_n\}$ and $\{b_1,\ldots b_n\}$ is an optimal solution

Running time: Sorting takes O(nlog n). Hence the running time of our algorithm is O(nlog n)

5) 15 pts

Given a connected graph G = (V,E) with positive edge weights and two nodes s,t in V, prove or disprove:

a. If all edge weights are unique, then there is a single shortest path between any two nodes in V.
b. If each edge's weight is increased by *k,* the shortest path cost will increase by a multiple of *k*.
c. If the weight of some edge *e* decreases by *k*, then the shortest path cost will decrease by at most *k*.

a) False. Counter Example: (s,a) with weight 1, (a,t) with weight 2 and (s,t) with weight 3. There are two shortest path from s to t though the edge weights are unique.

b) False. Example: suppose the shortest path *s→t* consist of two edges, each with cost 1, and there is also an edge *e=(s,t)* in G with cost(e)=3. If now we increase the cost of each edge by 2*, e* will become the shortest path (with the total cost of 5).

c) True. For any two nodes s,t, assume that $P_1,\ldots,P_k$ are all the paths from s to t. If e belongs to P_i then the path cost decrease by k, otherwise the path cost unchanged. Hence all paths from s to t will decrease by at most k. As shortest path is among them, then the shortest path cost will decrease by at most k.

Grading policy: 5 points for each item. 2 points for TRUE/FALSE part and 3 points for prove/disprove part.

6) 20 pts
Sam is shocked to find out that his word processor has corrupted his text document and has eliminated all spacing between words and all punctuation so his final term paper looks something like: "anawardwinningalgorithmto…". Luckily he has access to a Boolean function *dictionary( )* that takes a string *s* and returns true if *s* is a valid word and false otherwise. Considering that he has limited time to turn in his paper before the due date, find an algorithm that reconstructs his paper into a sequence of valid words with no more than O(n^2) calls to the *dictionary( )* function.

Solution: We denote that the string as s[1],….,s[n]. For the sub string s[1],….,s[m], we construct the table P as follows: if s[1],….,s[m] is composed of a sequence of valid words, then P[m]=true; otherwise P[m]=false. The recursive relation is as follows:

$$P[0] = true$$

$$P[m] = \vee_{0 \le i \le m-1} (P[i] \wedge dictionary\ \ (s[i+1],\dots,\ s[m]))$$

If P[m] is true, we denote q[m]=j where j is such that $P[j] \wedge dictionary(s[j+1],\dots,s[m])$ is true. By the definition of P[m], it is obvious that such a j must exist when P[m] is true. To reconstruct the paper into a sequence of valid words, we just need to output n, q[n],q[q[n]],…,0 as segmentation points.

Running time: The length of table of P and q is n and each step when we compute P[m] we need at most m calls of function dictionary(). Note the $m \le n$. Hence the running time is bounded by O(n^2)

Remark: Actually this question is exactly the same as the first problem in HW4: Problem 5 in Chapter 6 if we set $quality(Y_{i+1}, k) = 0$ if $dictionary(Y_{i+1}k) = \ false$ and $quality(Y_{i+1}, k) = 1$ if $dictionary(Y_{i+1}k) = true$.

7) 10 pts

An $n \times n$ array A consists of 1's and 0's such that, in any row of A, all the 1's come before any 0's in that row. Give the most efficient algorithm you can for counting the number of 1's in A.

Algorithm: For each row i, we use binary search to find the index of last element of 1 in the array $A_i[1,\ldots.n]$. Suppose the index is a_i. Then the sum of $a_i(0<i<n+1)$ is the number of 1's in A.

Proof: Assume that in i-th row k is the index of last element of 1. Then we have the property that $a_{ij}=1$ for $j<k+1$ and $a_{ij}=0$ for $j>k$ as all 1's come before any 0's in each row. During the binary search, if $a_{im}=1$, then we must have $k\geq m$; $a_{im}=0$, then we must have $k<m$. By doing this, we can finally find the value of k for each row.

Running time: There are n rows and for each row the running time to find the last element which is 1 by binary search is log n. Hence the running time of our algorithm is O(nlog n)

Grading policy: 5 points for algorithm. 3 points for proof and 2 points for running time.

CS570
Analysis of Algorithms
Summer 2007
Exam 1

Name: ________________________
Student ID: ____________________

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

1) 20 pts
Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification.

[TRUE/FALSE]
Given a connected graph G, with at least two edges having the same cost, there will definitely be two distinct minimum spanning trees.

[TRUE/FALSE]
You can use Dijikstra's shortest path algoritm in a graph with negative edge weights.

[TRUE/FALSE]
Prim's algorithm can be implemented in O(m log n) time.

[TRUE/FALSE]
In a dynamic programming problem, the solution to a sub-problem might change at certain steps and therefore has to be recomputed.

[TRUE/FALSE]
In the Gale-Shapley algorithm, the order in which the men propose to the women (i.e the order in which the algorithm might pick a man to propose to a woman) might affect the final matching that is formed.

[TRUE/FALSE]
if f(n) = Theta (g(n)), then g(n) = Theta (f(n)).

[TRUE/FALSE]
Given a graph G with unit edge costs, we can use the BFS algorithm to calculate the shortest path between two nodes.

[TRUE/FALSE]
A Pseudo-polynomial algorithm will always be inefficient, no matter what the magnitude of the input.

[TRUE/FALSE]
The shortest path between two nodes that are both on the minimum spanning tree consists only of those edges that are in the minimum spanning tree.

[TRUE/FALSE]
A Breadth First Search Tree on graph G can be used to determine distances between all nodes in G.

2) 10 pts
Fill in the following table. Each box should have a "yes" or "no" indicating whether or not the condition at the top of the row is true. The first row has already been answered for you. Basically, for each box you should ask yourself "Is B and upper/lower/asymptotic bound on A?"

A	B	A=O(B)	A=Ω(B)	A=Θ(B)
n	1			
n^2	n lgn			
$n+n^2$	n^3			
n	200n + n/2			
$n^{.5}$	lg n			
$n^{10^{10}}$	2^n			

3) 10 pts
What is the big-O running time, in terms of n, of the following code fragment?

a)
```
for (i= 1; i < n; i = i*2)
Sum++;
```

b)
```
int fact (int n) {
if (n == 1) return 1;
else return n*fact(n-1); }
```

c)
```
int silly (int n) {
if (n == 0) return 0;
else return 1 + silly(n-1) + silly(n-1);}
```

4) 15 pts

Suppose that you are given a strongly connected directed graph G (V,E) with positive edge weights and let $v_0 \in V$. You want to find the shortest paths between all pairs of nodes; however, you are only interested in paths that go through the particular node v_0. Present an efficient algorithm that solves this problem.

5) 20 pts
   Assume that you have a complete binary tree with values v_i, on each node. Starting at the root, you have the choice of picking up the value at a node. If you pick up the value at a node x, say v_x, you cannot pick up the values at any of the children of x. Give a dynamic program to calculate which nodes to pick maximizing the sum of the values of the nodes picked.

6) 15 pts
The object of the Coin Problem is to come up with the minimum number of coins to pay X cents.

a) Consider the greedy approach to solving the coin problem for US coins where we start with the largest coin (25¢ coin) and use it as many times as possible, then use as many 10¢ coins as possible on the remainder, then 5¢, then 1¢. Prove or disprove that "this greedy algorithm always gives an optimal solution", i.e. gives the minimum number of coins to pay X cents.

b) Now suppose that 5¢ coins are not allowed, only 1¢, 10¢, and 25¢. Prove or disprove that “the corresponding greedy approach (25¢ then 10¢ then 1¢) always gives an optimal solution”.

7) 10 pts
Given a connected graph G(V,E), an edge e is called an *isthmus* if the removal of e from G disconnects G. Prove that every spanning tree of G must contain all the isthmus edges.

CS570
Analysis of Algorithms
Fall 2007
Exam I

Name: ______________________

Student ID: __________________

	Maximum	Received
Problem 1	20	
Problem 2	20	
Problem 3	12	
Problem 4	12	
Problem 5	12	
Problem 6	12	
Problem 7	12	

Note: The exam is closed book closed notes.

1) 20 pts
Mark the following statements as **TRUE**, **FALSE**. No need to provide any justification.

[TRUE/FALSE]
A greedy algorithm is any algorithm that follows the heuristic of making the locally optimum choice at each stage with the hope of finding the global optimum.
T
[TRUE/FALSE]
BFS can be used to find the shortest path between any two nodes in a weighted graph.
F
[TRUE/FALSE]
DFS can be used to find the shortest path between any two nodes in a non-weighted graph.
F
[TRUE/FALSE]
BFS can be used to test whether a graph is bipartite
T
[TRUE/FALSE]
If T is a spanning tree of G and e an edge in G which is not in T, then the graph T+e has a unique cycle.
T
[TRUE/FALSE]
Let T be a spanning tree of a graph G and e an edge of G which is not in T. For any edge f that is on a cycle in graph T+e, the graph T + e − f is a spanning tree.
T
[TRUE/FALSE]
Given a graph G(V,E) with distinct costs on edges and a set $S \subseteq V$, let (u, v) be an edge such that (u, v) is the minimum cost edge between any vertex in S and any vertex in V-S. Then, the minimum spanning tree of G must include the edge (u, v).
T
[TRUE/FALSE]
If f, g, and h are positive increasing functions with f in O(h) and g in Ω(h), then the function f+g must be in Θ(h).
F
[TRUE/FALSE]
If a divide and conquer algorithm divides the problem is half at every step, the log(n) factor related to the depth of the recursion tree will cause the algorithm to have a lower bound of Ω(n log(n))
F
[TRUE/FALSE]
Suppose that in an instance of the original Stable Marriage problem with n couples, there is a man M who is last on every woman's list and a woman W who is last on every man's list. If the Gale-Shapley algorithm is run on this instance, then M and W will be paired with each other.
T

2) 20 pts

a) Arrange the following in the order of big oh $4n^2$, $\log_2(n)$, $20n$, 2, $\log_3(n)$, n^n, 3^n, $n\log(n)$, $2n$, 2^{n+1}, $\log(n!)$

$2<\log_2(n)< \log_3(n)<2n<20n< \log(n!) <n\log(n)< 4n^2< 2^{n+1}<3^n< n^n$

b) Find the complexity of the following nested loop

```
sum = 0;
for (i=0; i<3; i++)
  for (j=0; j<n; j++)
     sum++;
```

O(n)

c) Find the complexity of the following code section

```
for (i=0; i<n; i++) {
  for (j=0; j<n; j++) {
     A[i] = random(n);
   } // assume random() is O(1)
  sort(A, n); // assume sort() is the fastest general sorting algorithm
}
```

$O(n^2)$

d) Find the complexity of the following function

```
int somefunc(int n) {
   if (n <= 1)
      return 1;
   else
      return somefunc(n-1) + somefunc(n-1);
}
```

$O(2^n)$

e) Find the runtime T(n) in the following recurrence equation:
$T(n) = 2T(n/4)+ n^{0.51}$

$T(n)=O(n^{0.51})$

3) 12 pts

Suppose we are given an instance of the Shortest Path problem with source vertex s on a directed graph G. Assume that all edges costs are positive and distinct. Let P be a minimum cost path from s to t. Now suppose that we replace each edge cost c_e by its square, c_e^2, thereby creating a new instance of the problem with the same graph but different costs.

Prove or disprove: P still a minimum-cost s - t path for this new instance.

The statement is FALSE
Consider the example:
Vertices: V={A, B, C, D}
Edges: (A->B)=100, (A->C)=51, (B->D)=1, (C->D)=51

Shortest path from A to D is A->B->D Path length=101
After squaring this path length become 100^2+1^2=10001
However, A->C->D has path length 51^2+51^2=5202<10001
Thus A->C->D become shortest path from A to D

4) 12 pts
Consider a long country road with houses scattered very sparsely along it. You want to place cell phone base stations at certain points along the road, so that every house is within four miles of one of the base stations. Give an efficient algorithm that achieves this goal, using as few base stations as possible.

Greedy approach
1) Start from the beginning of the road
2) Find the first uncovered house on the road
3) If there is no such a house, terminate this algorithm; otherwise, go to next line
4) Locate a base station at 4 miles away after you find this house along the road
5) Go to 2)

Proof:
Let the number of base stations used to cover the first n houses in our algorithm be G(n). For any other policy, denote the number of base stations used to cover the first n houses P(n). Optimality of our algorithm can be shown if G(n)<=P(n) for n=1, 2, 3, ….We prove this by induction
It is obvious that G(1)<=P(1)
Given G(n-1)<=P(n-1), let's consider G(n) and P(n)
If the n-th house is already covered by G(n-1) base stations, then G(n)=G(n-1)<=P(n-1)<=P(n). This completes the proof.
If the n-th house has not been covered by G(n-1) base stations, then G(n)=G(n-1)+1. If G(n-1)<P(n-1), then we have P(n)>=P(n-1)>=G(n-1)+1=G(n), which completes the proof.
Otherwise G(n-1)=P(n-1). In this case the n-th house must have note been covered by P(n-1) base stations in this other policy because we always make sure that our policy covers the longest distance from the beginning to the n-th base station. Thus P(n)=P(n-1)+1>=G(n). This completes the proof.

Running time O(n), where n is the number of houses

5) 12 pts
Given a sorted array *A* of distinct integers, describe an algorithm that finds *i* such that *A*[*i*] = *i*, if such an *i* exists. Your algorithm must have a complexity better than *O(n)*.

```
Function(A,n)
        {
        i=floor(n/2)
        if A[i]==i
                return TRUE
        if (n==1)&&(A[i]!=i)
                return FALSE
        if A[i]<i
                return Function(A[i+1:n], n-i)
        if A[i>i]
                return Function(A[1:i], i)
        }
```

Proof:
The algorithm is based on Divide and Conquer. Every time we break the array into two halves. If the middle element i satisfy A[i]<j, we can see that for all j<i, A[j]<j. This is because A is a sorted array of DISTINCT integers. To see this we note that A[j+1]-A[j]>=1 for all j. Thus in the next round of search we only need to focus on A[i+1:n]
Likewise, if A[i]>i we only need to search A[1:i] in the next round.
For complexity T(n)=T(n/2)+O(1)
Thus T(n)=O(log n)

6) 12 pts

Consider a max-heap implemented using pointers rather than an array, so that the root has pointers to two smaller heaps, all of whose elements are smaller than the root's element. Give an algorithm to find the smallest element in the heap, and argue that your algorithm always runs in O(n) time on a heap with n elements.

Min=value at the root
Run a BFS or DFG through the heap. Every time a new node is visited compare its value with Min, if it is smaller then Min=this value

Every node visited only once, thus O(n)

CS570

Analysis of Algorithms

Spring 2008

Exam I

Name: ________________________

Student ID: ___________________

Section:__2:00-5:00 or __5:00-8:00

	Maximum	Received
Problem 1	20	
Problem 2	16	
Problem 3	12	
Problem 4	16	
Problem 5	16	
Problem 6	20	
Total	100	

Note: The exam is closed book closed notes.

1) 20 pts
Mark the following statements as **TRUE**, **FALSE**. No need to provide any justification.

[TRUE/FALSE]
Given the shortest path P between two nodes A and B in graph G(V;E), there exists a minimum spanning tree T of G, such that all edges of path P is contained in T.

[TRUE/FALSE]
If in a connected graph all edge weights are distinct, then for any two nodes A and B there exists a unique shortest path.

[TRUE/FALSE]
O(E log E) and O(E log V) are equivalent regardless whether graph is dense or sparse.

[TRUE/FALSE]
$2^n = \Theta(2^{n+4})$

[TRUE/FALSE]
Suppose $T_1(N) = O(F(N))$ and $T_2(N) = O(F(N))$. Then, $T_1(N) = O(T_2(N))$

[TRUE/FALSE]
Suppose that in an instance of the original Stable Marriage problem with n couples (so that every man ranks every woman and vice versa), there is a man M who is last on each woman's list and a woman W who is last on every man's list. If the Gale-Shapley algorithm is run on this instance, than M and W will be paired with each other.

[TRUE/FALSE]
DFS takes O(|E| + |V |) time if the graph is represented as an adjacency matrix. |E| is the number of edges and |V | is the number of vertices(nodes).

[TRUE/FALSE]
In a nested loop, if O(n) is a tight upper bound to the number of iterations in the outer loop and O(m) is a tight upper bound to the number of iterations in the inner loop, then O(mn) must be a tight upper bound on the number of times the inner loop is executed.

[TRUE/FALSE]
In an algorithm that uses a priority queue, the worst case complexity of the algorithm can be dependent on the type heap the priority queue is implemented with

[TRUE/FALSE]
Given a connected graph G, with at least two edges having the same cost, there will be at least two distinct minimum spanning trees

2) 16 pts

What is the worst case complexity, in terms of n, of the following code fragment

a-

```
for (i = 1; i<n;  i + +)
    for (j = i + 1; j < n; j + +)
        s + +;
        k = 2 * (s - 1);
    endfor
endfor
```

b-

```
for (i = 1; i < n; i + +)
    for (j = 1; j <i**2; j + +)
        s + +;
        k = 2 _ (s _ 1);
    endfor
endfor
```

c- Consider the following pseudocode, where A is an array storing n integer values. What are the best and `worst case time complexities of the algorithm AddAndAdd?

```
Algorithm AddAndAdd
c = 0
i = 0
while c < 200 and i < n do
    c = c+A[i]
    i=i+1
    for k = 0 to n – 1
        c = c + A[k]
    endfor
endwhile
```

3) 12 pts

Indicate for each pair of expressions (A, B) in the table below, where A is O, Ω, or θ of B. Assume that k and c are positive constants. Mark each box with Y (yes) and N (no).

A	B	O	Ω	θ
n^k	C^n			
2^n	$2^{n/2}$			
lg(n!)	$\lg(n_n)$			

4) 16 pts

Give an algorithm that takes a set of elements A[1] through A[n] and returns the maximum sum without neighbors. In other words, it computes the largest sum of elements in the array none of which are neighbors. (subsequent elements A(i) and A(i+1) are considered neighbors). The algorithm should run in polynomial time with respect to n.

5) 16 pts

There are three types of operations you can perform on an integer:

- If it’s divisible by 3, divide it by 3.

- If it’s divisible by 2, divide it by 2.

- Subtract 1.

Given an integer n, return the minimal number of operations needed to produce the number 1.

a) 20 pts
Suppose that you have a very old MP3 player. Songs cannot be shuffled, or organized by song-name. Instead, it lets users define a display-order for artists (they need not be alphabetical). All songs by a given artist will appear consecutively. The device scrolls down one song on its list when you whack it firmly on concrete or other hard surface. Another single button lets you reset to the top of the list. Thus, when you want to listen to songs by a particular group *X*, you push the reset, and then whack it enough times to scroll past all songs of all artists appearing before group *X*. Because it has an expected lifetime of only 20,000 whacks, you want to organize the artists so that on average it does not get whacked any more than necessary. You can formalize the problem this way: For each of n artists numbered $i = 1$ through n, a value s_i gives the number of songs stored for artist i, and a fraction $0 < f_i < 1$ for each i gives the frequency with which you intend to scroll to listen to songs by artist i. For example, see the following.

Artist	Number of songs	Access Frequency
A	15	1/4
B	8	1/3
C	12	1/6
D	5	1/4

Then, the average whacks needed to scroll to a desired artist's songs with the ordering (A,B,C,D) is $1/4\cdot0+1/3\cdot15+1/6\cdot23+1/4\cdot35$. The average whacks for the reverse ordering (D,C,B,A) would be $1/4\cdot0+1/6\cdot5+1/3\cdot17+1/4\cdot25$.

Describe an algorithm that, given a collection of artists 1, 2, … n, number of songs s_i for each, and frequency of access f_i for each, outputs an ordering of artists so that the average number of whacks is minimized.

CS570
Analysis of Algorithms
Summer 2008
Exam I

Name: ________________________
Student ID: ____________________

____4:00 - 5:40 Section ____6:00 – 7:40 Section

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

2 hr exam
Close book and notes

1) 20 pts

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

[TRUE/FALSE]

There exist a perfect matching and a corresponding preference list such that every man is part of an instability, and every woman is part of an instability.

FALSE

[TRUE/FALSE]

A greedy algorithm always returns the optimal solution.

FALSE

[TRUE/FALSE]

The function 100n + 3 is O(n^2)

FALSE

[TRUE/FALSE]

You are given n elements to insert into an empty heap. You could directly call heap insert n times. Or you could first sort the elements and then call heap insert n times. In either case, the asymptotic time complexity is the same.

TRUE

[TRUE/FALSE]

If a problem can be solved correctly using the greedy strategy, there will only be one greedy choice (e.g. "choose the object with highest value to weight ratio") for that problem that leads to the optimal solution.

FALSE

[TRUE/FALSE]

The Depth First Search algorithm can not be used to test whether a directed graph is strongly-connected.

FALSE

[TRUE/FALSE]

Consider an undirected graph G=(V, E) with non-negative edge weights. Suppose all edge weights are different. Then the edge of maximum weight can be in the minimum spanning tree.

TRUE

[TRUE/FALSE]

Consider a perfect matching S where Mike is matched to Susan. Suppose Mike prefers Winona to Rachel. Then the pair (Mike, Rachel) can never be an instability with respect to S.

FALSE

[TRUE/FALSE]
Consider an undirected graph G=(V, E). Suppose all edge weights are different. Then the shortest path from A to B must be unique.

FALSE

[TRUE/FALSE]
The number of elements in a heap must always be an integer power of 2.

FALSE

2) 20 pts
Given two graphs G and G' that have the same sets of vertices V and edges E, however different weight functions (W and W' respectively) on their edges. Suppose for each graph the weights on the edges are distinct and satisfy the following relation: $W'(e)=W(e)^2$ (Note: all edge weights in G and G' are positive integers) for every edge e of E. Decide whether each of the following statements is true. Give either a short proof or a counter example.

a) The minimum spanning tree of G is the same as the minimum spanning tree of G'.

<u>Solution</u>

This statement is true

Now, since the edges costs are all distinct, the order of the sorted lists of the edges will be the same in G and G'. This is because if a > b , then $a^2>b^{2.}$

Now, if we run, Prims algorithm, it will consider the edges in the same order in the case of both G and G'. Therefore, since the structure of G and G' are same and also the ordering of the edges (with respect to edge cost) is the same, the same MST will be produced.

b) For a pair of vertices *a* and *b* in V, a shortest path between them in G is also a shortest path in G'.

<u>Solution</u>

This statement is false. Hint : Pythagoras's theorem

3) 10 pts
Prove or give a counterexample: An array that is sorted in ascending order is a min-heap.

<u>Solution</u>

Let's assume that the statement is false. Suppose H is a binary tree for the array sorted in ascending order. By the statement, for every element v at a node i in H,
1. the element w at i's parent satisfies key(w) $\leq$ key(v).
2. the element w at node i and its right sibling w' satisfies key(w) $\leq$ key(w').

Min-heap property: for every element v at a node i and the element w at i's parent satisfies key(w) $\leq$ key(v).

H satisfies the min-heap property. Contradiction. Thus, the statement is true.

4) 15 pts
Let G = (V,E) be an undirected graph with maximum degree d. A coloring of G is an assignment to each vertex of G of a "color" such that adjacent vertices have distinct colors. Consider the greedy algorithm that colors as many vertices as possible with color "j" before moving to color "j+1."
Prove or give a counterexample: This greedy algorithm never requires more than d+1 colors to color G.

<u>Solution:</u>

Proof: Let χ(G) denotes number of colors this greedy algorithm requires to color G. Prove by contradiction.
Assume there exists a graph G' with maximum degree d such that χ(G')>=d+2. Let *v* be the first vertex in G' colored with color "d+2" by the algorithm. Hence all vertices adjacent to *v* must have used up color "1" through "d+1" (otherwise by the algorithm color "d+2" would not be used), which means *v* has at least d+1 adjacent vertices, that is, the degree of v is at least d+1. But the maximum degree in G' is d. Contradiction. Therefore, this greedy algorithm never requires more than d+1 colors to color G.

5) 10 pts
You and your eight-year-old nephew Shrek decide to play a simple card game. At the

beginning of the game, the cards are dealt face up in a long row. Each card is worth a different number of points. After all the cards are dealt, you and Shrek take turns removing either the leftmost or rightmost card from the row, until all the cards are gone. At each turn, you can decide which of the two cards to take. The winner of the game is the player that has collected the most points when the game ends.
Having never taken an algorithms class, Shrek follows the obvious greedy strategy—when it's his turn, Shrek always takes the card with the higher point value. Your task is to find a strategy that will beat Shrek whenever possible. (It might seem mean to beat up on a little kid like this, but Shrek absolutely hates it when grown-ups let him win.)
Prove that you should not also use the greedy strategy. That is, show that there is a game that you can win, but only if you do not follow the same greedy strategy as Shrek.

Solution

Let S be shrek's total point and S' be my total point at ith turn. Let p_k, p_{k+1}, p_{k+2}, p_{k+3}, p_{k+4}, p_{k+5} be the sums of each pair of cards that remained, where k >= 1 and $p_{k+2} >> p_{k+3} > p_{k+4} > p_{k+1} > p_{k+2}$. By the greedy strategy, the order that Shrek picks the pair of cards would be p_{k+4}, p_{k+2}, p_{k+1} and my order would be p_{k+3} then p_k. The total point at the final turn is that $S + p_{k+4} + p_{k+2} + p_{k+1} > S' + p_{k+3} + p_k$. If I don't use the greedy strategy, the result at the final turn would be $S + p_{k+4} + p_{k+3} + p_{k+1} < S' + p_k + p_{k+2}$. Contradiction. Thus, the greedy strategy is not an optimal solution.

6) 10 pts
Arrange the following functions in increasing order of asymptotic complexity. If $f(n)=\Theta(g(n))$ then put f=g. Else, if $f(n)=O(g(n))$, put f < g.
$4n^2$, $\log_2(n)$, $20n$, 2, $\log_3(n)$, n^n, 3^n, $n\log(n)$, $2n$, 2^{n+1}, $\log(n!)$

Solution: $2 < \log_2(n) = \log_3(n) < 2n = 20n < n\log(n) = \log(n!) < 4n^2 < 2^{n+1} < 3^n < n^n$

7) 15 pts
Prove or give a counterexample: Let G be an undirected, connected, bipartite, weighted graph. If the weight of each edge in G is +1, and for every pair of vertices (u,v) in G there is exactly one shortest path, then G is a tree.

Solution

The statement is true.

It is same as the following statement:
G has loop(s) => there exists at least one pair of vertices (u,v) in G with more than one shortest path.

G is a bipartite graph => it can not contain an odd cycle => all the cycle(s) have even number of nodes=> all the cycle(s) have length 2n (since edge weight is 1)

Take a smallest loop with 2m nodes and length 2m in G, then any pair of nodes (u,v) that are farthest away have two paths with equal length m. Also, since the loop is smallest, there doesn’t not exist any path <m between (u,v).

Proved.

CS570
Analysis of Algorithms
Fall 2008
Exam I

Name: ______________________
Student ID: __________________

____Monday Section ____Wednesday Section ____Friday Section

	Maximum	Received
Problem 1	20	
Problem 2	10	
Problem 3	10	
Problem 4	20	
Problem 5	20	
Problem 6	20	
Total	100	

2 hr exam
Close book and notes

1) 20 pts
Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification.

[TRUE/FALSE] F
Given graph *G* and a Minimum Spanning Tree *T* on *G*, you could find the (weighted) shortest path between arbitrary pair *u, v* in *V(G)* using only edges in *T*.

[TRUE/FALSE] F
$V^2 \log V = \theta (E \log E^2)$ whether the graph is dense or sparse.

[TRUE/FALSE] T
If DFS and BFS returns different trees, then the original graph is not a tree.

[TRUE/FALSE] T
An algorithm with the running time of $n * 2^{\min(n \log n, 10000)}$ runs in polynomial time.

[TRUE/FALSE] T
We may need to run Dijkstra’s algorithm to compute the shortest path on a directed graph, even if the graph doesn't have a cycle.

[TRUE/FALSE] F
Given a graph that contains negative edge weights, we can use Dijkstra's algorithm to find the shortest paths between any two vertexes by first adding a constant weight to all of the edges to eliminate the negative weights.

[TRUE/FALSE] F
If $f(n)$ and $g(n)$ are asymptotically positive functions, then
$f(n)+g(n) = \theta(\min\{f(n),g(n)\})$.

[TRUE/FALSE] F
For a stable matching problem such that *m* ranks *w* last and *w* ranks *m* last, *m* and *w* will never be paired in a stable matching.

[TRUE/FALSE] F
Breadth first search is an example of a divide-and-conquer algorithm.

[TRUE/FALSE] T
Kruskal's algorithm for finding the MST works with positive and negative edge weights.

2) 10 pts

a) Arrange the following in the increasing order of asymptotic growth. Identify any ties.

$lg\ n^{10},\ 3^n,\ lg\ n^{2n},\ 3n^2,\ lg\ n^{lg\ n},\ 10^{lg\ n},\ n^{lg\ n},\ n\ lg\ n$

$lg\ n^{10},\ lg\ n^{lg\ n},\ lg\ n^{2n},\ n\ lg\ n,\ 3n^2,\ 10^{lg\ n},\ n^{lg\ n},\ 3^n$

b) Analyze the complexity of the following loops:

```
i- x = 0
   for i=1 to n
       x= x + lg n
   end for
```

O(n)

```
ii- x=0
    for i=1 to n
         for j=1 to lg n
             x = x * lg n
         endfor
     endfor
```

O(nlgn)

```
iii- x = 0
     k = “some constant”
     for i=1 to max (n, k)
         x= x + lg n
     end for
```

O(n)

```
iv- x=0
    k = “some constant”
    for i=1 to min(n, k)
         for j=1 to lg n
             x = x * lg n
         endfor
    endfor
```

O(lgn)

3) 10 pts

a) For each of the following recurrences, give an expression for the runtime T (n) if the recurrence can be solved with the Master Theorem. Otherwise, indicate that the Master Theorem does not apply.

$T(n) = T(n/2) + 2^n$

$\theta(2^n)$

$T(n) = 16T(n/4) + n$

$\theta(n^2)$

$T(n) = 2T(n/2) + n \log n$

Master Theorem does not apply.

$T(n) = 2T(n/2) + \log n$

$\theta(n)$

$T(n) = 64T(n/8) - n^2 \log n$

Master Theorem does not apply.

b) Suppose we have a divide and conquer algorithm which at each step takes time n, and breaks the problem up into $n^{1/2}$ subproblems of size $n^{1/2}$ each. Find the runtime $f(n)$ such that $T(n) = \theta(f(n))$, given the following recurrence. $T(n) = n^{1/2}\, T(n^{1/2}) + n$

They have to show the full recursion tree.
At the root we spend cn
At the next level we have $n^{1/2}$ nodes each taking c$n^{1/2}$ so again, at this level we spend cn time
Same goes with every other level. There are lgn levels in the tree because at every we take the square root of n. So the total run time is $\theta(nlgn)$

4) 20 pts
A key to customer service is to avoid having the customer wait longer than necessary. That's your philosophy anyway when you open a coffee shop shortly after graduation. This philosophy and your CS background have made you wonder about the order in which your server should serve customers whose orders have different levels of complexity. For example, if customer one's order will take 3 minutes while customer two's will take 1 minute, then serving customer one first results in 7 minutes of waiting (3 for customer one and 4 for customer two) while serving customer two first results in only 5 (1 for customer two and 4 for customer one).
(a) Phrase this problem more formally by introducing notation for the service time and precisely define the problem's objective.

Please refer to solution to Q4

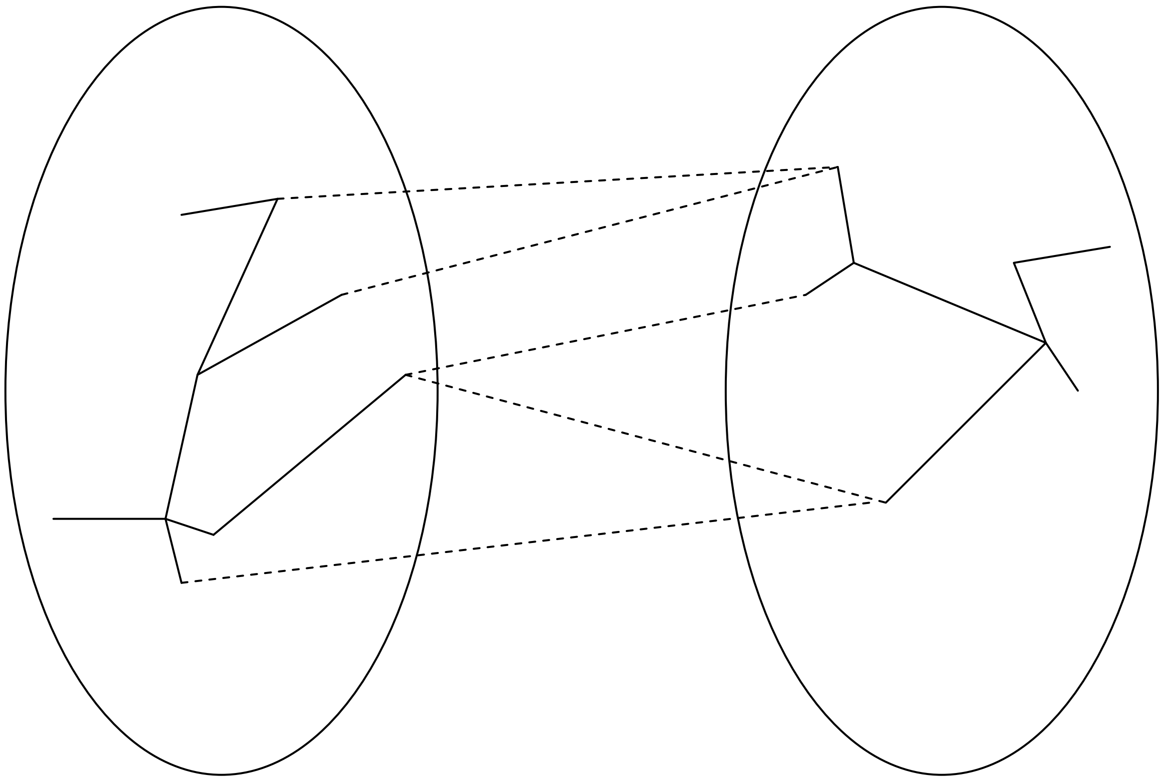
(b) Give a greedy algorithm and use an interchange argument to prove that it works for a single server as long as no new customers arrive.

(c) Show how your algorithm can fail if new customers arrive while the others are being served. (For this problem, assume that you cannot stop serving a customer once you start; coffee equipment is not designed for context switching.)

(d) Show that your algorithm does not necessarily find the best solution if the coffee shop has two servers.

5) 20 pts

Assume we have two graphs G1 = (V1, E1) and G2 = (V2, E2). Also assume that we have T1 which is a MST of G1 and T2 which is MST of G2. Now consider a new graph G = (V, E) such that V = V1 ∪ V2 and E = E1 ∪ E2 ∪ E3 where E3 is a new set of edges that all cross the cut (V1, V2). Following is an example of what G might look like.



The dashed edges are E3, the solid edges in G1 (on the left) are T1, and the solid edges in G2 (on the right) are T2.

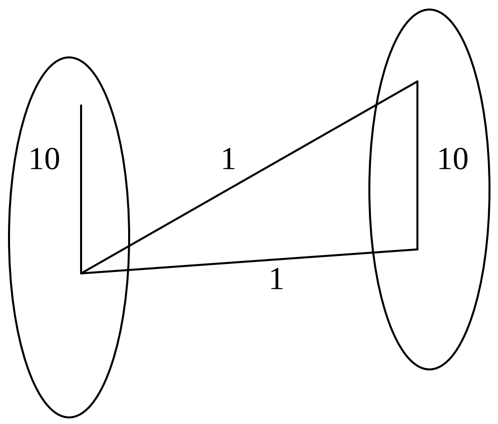
Now assume we want to fine a MST of the new graph. Consider the following algorithm:

```
Maybe-MST(T1, T2, E3)
e_min =  a minimum weight edge in E3
T = T1 U T2 U { e_min}
return T
```

Prove or disprove this algorithm (space provided on next page)

Additional space

Can show a simple counter example.



6) 20 pts
For an unsorted array x of size n, give a divide and conquer algorithm that runs in *O(n)* time and computes the maximum sum *M* of two adjacent elements in an array.
$M = Max\ (x_i + x_{i+1});\ i=1 \text{ to } n-1$

Divide: divide problem into 2 equal size subproblems at each step
Conquer: solve the subproblems recursively until the subproblem become trivial to solve. In this case problem size of 2 is trivial. In that case the solution is the sum of the 2 numbers.
Combine: We need to merge the solutions to the two subproblems S1 and S2. At each step we need to evaluate the following 3 cases:
Case 1: Max is in S1 (subproblem to the left)
Case 2: Max is in S2 (subproblem to the right)
Cases 3: Max is on the border of S1 and S2
To achieve this. In addition to finding the Max and sending it up the recursion tree at each step we need to send the two numbers at the ends of the subarrays S1 and S2 at each step. So using the following convention:
F1 = first element of S1
L1 = last element of S1
MAX1 = maximum sum of two adjacent elements in S1
F2 = first element of S2
L2 = last element of S2
MAX2 = maximum sum of two adjacent elements in S2

Then we will combine the two subproblems as follows:

```
If (L1+F2 > MAX1 and L1+F2 > MAX2 ) then
    F = F1
    L = L2
    MAX = L1+F2
Else if (MAX1 > MAX2) then
    F = F1
    L = L2
    MAX = MAX1
Else
    F = F1
    L = L2
    MAX = MAX2
endif
```

CS570
Analysis of Algorithms
Spring 2009
Exam I- Solution

1) 20 pts
Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification.

[TRUE/FALSE]
Given a weighted graph and two nodes, it is possible to list all shortest paths between these two nodes in polynomial time.
False. Not valid for graph with negative weights.

[TRUE/FALSE]
Given a graph G = (V, E) with cost on edges and a set S which is a subset of V, let (u,v) be an edge such that (u, v) is the minimum cost edge between any vertex in S and any vertex in V - S. Then, the minimum spanning tree of G must include the edge (u, v).

True. Refer to textbook, page 145, theorem 4.17.

[TRUE/FALSE]
Let G = (V, E) be a weighted graph and let M be a minimum spanning tree of G. The path in M between any pair of vertices v1 and v2 must be a shortest path in G.

False. Counter example: a loop with three nodes A, B and C. The edge costs are A-B: 1, B-C: 1, A-C: 1.5. Edge A-C is not included in M, but the shortest path between A and C is through edge A-C.

[TRUE/FALSE]
$2n^2 + \theta(n) = \theta(n^2)$

True.

[TRUE/FALSE]
Kruskal's algorithm can fail in the presence of negative cost edges.

This is False.

[TRUE/FALSE]

For any cycle in a graph, the cheapest edge in the cycle is in a minimum spanning tree.
False. If every edge in a cycle C is very expensive then it is possible that the MST doesn't use any edge in C (including the cheapest edge in C).

[TRUE/FALSE]
In every instance of the stable marriage problem there is a stable matching containing a pair (m, w) such that m is ranked first on the preference list of w and w is ranked first on the preference list of m.

False. Consider the following scenario where there are two men and women each in M and W with the following preferences:

m prefers w to w'

m' prefers w' to w

w prefers m' to m

w' prefers m to m'

[TRUE/FALSE]
Let T(n) be a function obeying the recurrence T(n) = 5T(n/5) + a with initial condition T(1) = b, where a and b are positive numbers. Then T(n) = Θ(n log n).
False. By the Master Theorem T(n) = Θ(n). We compare $n\log_5^5$ with the overhead term of a = $O(n^0)$ and find that we are in the case where the former dominates.

[TRUE/FALSE]
If the edges in a connected undirected graph are unit cost, then you can find the MST using BFS.

This is true.
[TRUE/FALSE]
Asymptotically, a running time of n^d is better than $10n^d$

This is False. Asymptotically, both of them are the same Basically within theta of each other

2) 10 pts
   Give an algorithm that given as input a sequence of *n* numbers and a natural number *k* < *n*, outputs the smallest *k* numbers of the sequence in O(n + k log n) time.

Solution: Building a MIN-HEAP takes O(n) time and doing EXTRACT-MIN k times takes O(k log n) time and gives us the k smallest numbers of the sequence. Hence running time is O(n + k log n).

3) 10 pts
Arrange the following in the increasing order of asymptotic growth (x is a constant which is greater than 1).

$x^{\sqrt{\log n}}$ $\quad x^n$ $\quad n^{10}$ $\quad n(\log n)^4$ $\quad n^{\log n}$ $\quad x^{x^n}$ $\quad x^{n^x}$

Solution:

$x^{\sqrt{\log n}}$ $\quad n(\log n)^4$ $\quad n^{10}$ $\quad n^{\log n}$ $\quad x^n$ $\quad x^{n^x}$ $\quad x^{x^n}$

4) 20 pts

You have a sufficient supply of coins, and your goal is to be able to pay any amount using as few coins as possible.

(a) Describe a greedy algorithm to solve the problem when the coins usedare of values 1, 5, 10 and 25 cents. Prove the optimality of the algorithm.

Solution: Let k be the number of coins of denomination d_i just enough to be greater than equal to d_{i+1}, where d_{i+1} is the next higher denomination. Then k coins of denomination d_i can be replaced by d_{i+1} and some other coin(s) using less than k coins. One can verify this observation by taking into account the fact that 5 pennies can be replaced by a nickel, 2 nickels can be replaced by a dime and 3 dimes can be replaced by a quarter and a nickel (2 coins).

Let the money to change for be n, n = 25a + 10b + 5c + d. a, b, c, d are the numbers of quarters, dimes, nickels and pennies. We have b < 3, c < 2 and d < 5 (or it will violate the conclusion in the observation, e.g., if b$\geq$3, we can replace 3 dimes by a quarter and a nickel which results in less coins). We prove that when n $\geq$25, we should always pick a quarter. As 10b+5c+d $\leq$10*2+5*1+1*4 = 29, it means in the optimal solution, the amount of the money of the dimes, nickels and pennies will be no more than 29 cents. So we only need to prove the cases when n is 25, 26, 27, 28 or 29. One can easily verify that in such cases it is always better to choose a quarter. Similarly we can prove that when 10 $\leq$n < 25, we should always pick a dime and when 5 $\leq$n < 10 we should always pick a nickel.

(b) Show that a greedy algorithm does not yield optimal results if you are using only coins of value 1 cent, 7 cents and 10 cents.

Solution: Counter example: Consider making change for x = 14. The greedy algorithm yields 1× (10 cents) + 4× (1 cents). This is 5 coins. However, we can see that 2× (7 cents) = 14 = x also. Here we use 2 coins. Obviously 2 < 5, so Greedy is sub-optimal in the case of coin denominations 1, 7, and 10.

5) 10 pts
Let T be a minimum spanning tree for G with edge weights given by weight function w. Choose one edge (x, y) ∈ T and a positive number k, and define the weight function w’ by

w’(u, v) = w(u, v), if (u, v) ≠ (x, y)
w’(u, v) = w(u, v) – k, if (u, v) = (x, y)

Show that T is also a minimum spanning tree for G with edge weights given by w’.

Solution: Proof by contradiction. Note T is a spanning tree for G with weight function w’ and w’(T) = w(T) – K.

Suppose T is not a MST for G with w’ and hence there exists a tree T’ which is an MST for G with w’. So w’(T’) < w’(T) (*).

We have two cases to analyze. First, if (x, y)) Î T’ then w’(T’) = w(T’) – k and from the above we have w(T’) < w(T). Now T’ is a spanning tree for G with w which is better than T which is a MST for G with w. A contradiction. Case 2, suppose (x,y) Ï T’ then w’(T’) = w(T’) and so by (*) we have w(T’) < w’(T) = w(T) – k. A similar contradiction again.

Hence T is a MST for G with w’.

6) 15 pts

Stable Matching: Prove that, if all boys have the same list of preferences, and all the girls have the same list preferences, there can be only one stable marriage.

Solution: Proof by contradiction. Suppose there are n boys b1, b2, . . . , bn; and n girls g1, g2, . . . , gn. Without loss of generality, assume that all the boys prefer g1 over g2, g2 over g3, and so on. Likewise all the girls most prefer b1 and least prefer bn. It is easy to verify that the pairing (b1, g1), (b2, g2), . . . , (bn, gn) is a stable matching. Suppose that there is another stable matching, including couple (bi, gj) for i != j. If there are multiple such pairs, let i be the smallest such value. Therefore j > i, because g1 to gi－1 are engaged to b1 through bi－1 respectively: so j cannot be less than i. By the definition of the problem, everyone is engaged, including gi, and thus for some k > i, our alternate pairing includes the couple (bk, gi). Note that bi is a more popular boy than bk, since we chose i to be the smallest possible value. Therefore gi prefers bi over her fiance. And because i < j, we know bi prefers gi over his fiancee. Thus bi and gi constitute a instable couple. But this contradicts our supposition that this alternate matching is stable. So we conclude that there is no alternate stable matching.

7) 15 pts
Give a divide-and-conquer algorithm to find the average of n real numbers. You should clearly show all steps (divide, conquer, and combine) in your pseudo code. Analyze complexity of your solution.

Solution:

Divide: divide array up in 2 equal pieces
Conquer: solve the two subproblems reursively and return the average
Combine: Compute the average for the orginal problem by:
Average = (number of items in sub problem 1 * average for subproblem 1 + number of items in sub problem 2 * average for subproblem 2)/ (size of the original problem)

Complexity of Divide is O(1), complexity of Combine is O(1). Master theorem gives you a complexity of O(n) for the solution.

CS570
Analysis of Algorithms
Summer 2009
Exam I

Name: ________________________
Student ID: ____________________

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

2 hr exam
Close book and notes

1) 20 pts
Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification.

[TRUE/FALSE] T
An array with following sequence of terms [20, 15, 18, 7, 9, 5, 12, 3, 6, 2] is a max-heap.

[TRUE/FALSE] T
Complexity of the following shortest path algorithms are the same:
- Find shortest path between S and T
- Find shortest path between S and all other points in the graph

[TRUE/FALSE] T
In an undirected weighted graph with distinct edge weights, both the lightest and the second lightest edge are in the MST.

[TRUE/FALSE] F
Dijkstra's algorithm works correctly on graphs with negative-cost edges, as long as there are no negative-cost cycles in the graph.

[TRUE/FALSE] T
Not all recurrence relations can be solved by Master theorem.

[TRUE/FALSE] F
Mergesort does not need any additional memory space other than that held by the array being sorted.

[TRUE/FALSE] T
An algorithm with a complexity of $O(n^2)$ could run faster than one with complexity of $O(n)$ for a given problem.

[TRUE/FALSE] F
There are at least 2 distinct solutions to the stable matching problem--one that is preferred by men and one that is preferred by women.

[TRUE/FALSE] F
A divide and conquer algorithm has a minimum complexity of *O(n log n)* since the height of the recursion tree is always *O(log n)*.

[TRUE/FALSE] T
Stable matching algorithm presented in class is based on the greedy technique.

2) 10 pts

a) Arrange the following in the increasing order of asymptotic growth. Identify any ties.

$$lg\ n^{10},\ 3^n,\ lg\ n^{2n},\ 3n^2,\ lg\ n^{lg\ n},\ 10^{lg\ n},\ n^{lg\ n},\ n\ lg\ n$$

If lg is log2 (convention),

$$lg\ n^{10} < lg\ n^{2n} < lg\ n^{2n} = n\ lg\ n < 3n^2 < 10^{lg\ n} < n^{lg\ n} < 3^n$$

If lg is log10 (from ISO specification)

$$lg\ n^{10} < lg\ n^{2n} < 10^{lg\ n} < lg\ n^{2n} = n\ lg\ n < 3n^2 < n^{lg\ n} < 3^n$$

b) Analyze the complexity of the following loops:

```
i- x = 0
   for i=1 to n
       x= x + lg n
   end for
```

O(n)

```
ii- x=0
    for i=1 to n
         for j=1 to lg n
             x = x * lg n
         endfor
     endfor
```

O(nlogn)

```
iii- x = 0
     k = "some constant"
     for i=1 to max (n, k)
         x= x + lg n
     end for
```

O(n)

```
iv- x=0
    k = "some constant"
    for i=1 to min(n, k)
         for j=1 to lg n
             x = x * lg n
         endfor
    endfor
```

O(logn)

3) 10 pts

a) For each of the following recurrences, give an expression for the runtime T (n) if the recurrence can be solved with the Master Theorem. Otherwise, indicate that the Master Theorem does not apply.

$T(n) = T(n/2) + 2^n$

theta(2^n)

$T(n) = 16T(n/4) + n$

theta(n^2)

$T(n) = 2T(n/2) + n \log n$

More general case 2
If F(n) = theta((n^logb(a))*(logn)^k) with k >= 0, then T(n) = theta((n^logb(a))*(logn)^(k+1))

theta(n(logn)^2) from case 2

$T(n) = 2T(n/2) + \log n$

theta(n)

$T(n) = 64T(n/8) - n^2 \log n$
Does not apply (f(n) is not positive)

4) 20 pts

You work for a small manufacturing company and have recently been placed in charge of shipping items from the factory, where they are produced, to the warehouse, where they are stored. Every day the factory produces n items which we number from 1 to n in the order that they arrive at the loading dock to be shipped out. As the items arrive at the loading dock over the course of the day they must be packaged up into boxes and shipped out. Items are boxed up in contiguous groups according to their arrival order; for example, items 1… 6 might be placed in the first box, items 7…10 in the second, and 11… 42 in the third.

Items have two attributes, *value* and *weight*, and you know in advance the values *v1* … *vn* and weights *w1*…*wn* of the *n* items. There are two types of shipping options available to you:

Limited-Value Boxes: One of your shipping companies offers insurance on boxes and hence requires that any box shipped through them must contain no more than V units of value. Therefore, if you pack items into such a “limited-value” box, you can place as much weight in the box as you like, as long as the total value in the box is at most V.

Limited-Weight Boxes: Another of you shipping companies lacks the machinery to lift heavy boxes, and hence requires that any box shipped through them must contain no more than *W* units of weight. Therefore, if you pack items into such a “limited-weight” box, you can place as much value in the box as you like, as long as the total weight inside the box is at most *W*.

Please assume that every individual item has a value at most V and a weight at most W. You may choose different shipping options for different boxes. Your job is to determine the optimal way to partition the sequence of items into boxes with specified shipping options, so that shipping costs are minimized.

Suppose limited-value and limited-weight boxes each cost $1 to ship. Describe an *O(n)* greedy algorithm that can determine a minimum-cost set of boxes to use for shipping the *n* items. Justify why your algorithm produces an optimal solution.

We use a greedy algorithm that always attempts to pack the largest possible prefix of the remaining items that still fits into some box, either limited-value or limited weight. The algorithm scans over the items in sequence, maintaining a running count of the total value and total weight of the items encountered thus far. As long as the running value count is at most V or the running weight count is at most W, the items encountered thus far can be successfully packed into some type of box. Otherwise, if we reach a item j whose value and weight would cause our counts to <u>***exceed V and W***</u> , then prior to processing item j we first package up the items scanned thus far (up to item j-1) into an appropriate box and zero out both counters. Since the algorithm spends only a constant amount of work on each item, its running time is O(n).

Why does the greedy algorithm generate an optimal solution (minimizing the total number of boxes)? Suppose that it did not, and that there exists an optimal solution different from the greedy solution that uses fewer boxes. Consider, among all optimal solutions, one which agrees with the greedy solution in a maximal prefix of its boxes. Let us now examine the sequence of boxes produced by both solutions, and consider the first box where the greedy and optimal solutions differ. The greedy box includes items i…j and the optimal box includes items i…k, where k < j (since the greedy algorithm always places the maximum possible number of items into a box). In the optimal solution, let us now remove items k+1…j from the boxes in which they currently reside and place them in the box we are considering, so now it contains the same set of items as the corresponding greedy box. In so doing, we clearly still have a feasible packing of items into boxes and since the number of boxes has not changed, this must still be an optimal solution; however, it now agrees with the greedy solution in one more box, contradicting the fact that we started with an optimal solution agreeing maximally with the greedy solution.

5) 15 pts

For two unsorted arrays x and y, each of size n, give a divide and conquer algorithm that runs in *O(n)* time and computes the maximum sum *M,* as given below:

$M = Max\ (x_i + x_{i+1} - y_i);\ i=1 \text{ to } n-1$

Divide: divide problem into 2 equal size subproblems at each step.
Conquer: solve the subproblems recursively until the subproblem become trivial to solve. In this case problem size of 2 is trivial. In that case the solution is the sum of the 2 numbers.
Combine: We need to merge the solutions to the two subproblems S1 and S2. At each step we need to evaluate the following 3 cases:
Case 1: Max is in S1 (subproblem to the left)
Case 2: Max is in S2 (subproblem to the right)
Case 3: Max is on the border of S1 and S2

In case 3, you need to calculate X(the last element of S1) + X(the first element of S2) – Y(the last element of S1)

6) 10 pts

Suppose we are given an instance of the Shortest Path problem with source vertex s on a directed graph G. Assume that all edges costs are positive and distinct. Let P be a minimum cost path from s to t. Now suppose that we replace each edge cost c_e by its square, c_e^2, thereby creating a new instance of the problem with the same graph but different costs.
Prove or disprove: P still a minimum-cost s - t path for this new instance.

Let G have edges (s,v), (v,t), and (s,t), when the first two of these edges have cost 3 and the third has cost 5. Then the shortest path is the single edge (s,t), but after squaring the costs the shortest path would go thourgh v.

7) 15 pts

We are given two arrays of integers A[1..n] and B[1..n], and a number X. Design an algorithm which decides whether there exist i, j 2 {1, . . . , n} such that A[i] + B[j] = X. Your algorithm should run in time O(n log n).

```
 Sort array B
 For int i = 0 to n
    Temp = X – a[i]
    Do binary search to find an element whose value is equal to X – A[i] in array B
    If it finds return true
End of For
```

x

CS570
Analysis of Algorithms
Fall 2009
Exam I

Name: ______________________
Student ID: __________________

____Monday ____Friday 2-5 ____Friday 5-8 ____DEN

	Maximum	Received
Problem 1	20	
Problem 2	15	
Problem 3	15	
Problem 4	15	
Problem 5	15	
Problem 6	20	
Total	100	

2 hr exam
Close book and notes

If a description of an algorithm is required, please limit your description within 200 words, anything beyond 200 words will not be considered. Proof/justification or complexity analysis will only be considered if the algorithm is either correct or provided by the problem.

1) 20 pts

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

[TRUE]

Given a min-heap with *n* elements, the time complexity of select the smallest element from the *n* elements is O(1).

[FALSE] But we give credit to both true and false

Given a min-heap with *n* elements, the time complexity of select the second smallest element from the *n* elements is O(1).

[FALSE] Algorithms (e.g., bubble sort) with O(n^2) time complexity could cost O(n) in some instances.

Given a problem with input of size *n*, a solution with O(n) time complexity always costs less in computing time than a solution with O(n^2) time complexity.

[FALSE]

By using a heap, we can sort any array with *n* integers in O(n) time.

[TRUE] m is at most n(n-1)

For any graph with *n* vertices and *m* edges we can say that m = O(n^2).

[FALSE] m could be n(n-1)

For any graph with *n* vertices and *m* edges we can say that O(m+n) = O(m).

[FALSE] Consider the following counter-example:
G(V, E). V={A,B,C}, (A,B)=2, (A,C)=1, (B,C)=1. P=AB is the shortest path between A and B, but it is not in the MST.

Given the shortest path P between two nodes A and B in graph G(V,E), then there exists a minimum spanning tree T of G, such that all edges of path P is contained in T.

[FALSE] Consider the following counter-example:
w1 prefers m1 to m2; w2 prefers m2 to m1; m1 prefer w1 to w2; m2 prefer w2 to w1

Consider an instance of the Stable Matching Problem in which there exists a man *m* and a woman *w* such that *m* is ranked last on the preference list of *w* and *w* is ranked last on the preference list of *m*, then in every stable matching *S* for this instance, the pair (*w*, *m*) belongs to *S*.

[FALSE] A graph could have multiple MSTs.

While there are many algorithms to find the minimum spanning tree in a graph, they all produce the same minimum spanning tree.

[TRUE]

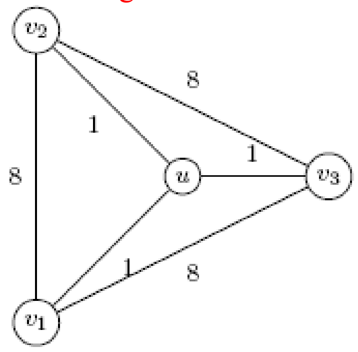
A BFS tree is a spanning tree

2) 15 pts

Here is a divide-and-conquer algorithm that aims at finding a minimum spanning tree. Given a graph $G = (V, E)$, partition the set V of vertices into two sets $V1$ and $V2$ such that $|V1|$ and $|V2|$ differ by at most 1. Let $E1$ be the set of edges that are incident only on vertices in $V1$, and let $E2$ be the set of edges that are incident only on vertices in $V2$. Recursively solve a minimum spanning tree problem on each of the two subgraphs $G1 = (V1, E1)$ and $G2 = (V2, E2)$. Finally, select the minimum-weight edge in E that crosses the cut $(V1, V2)$, and use this edge to unite the resulting two minimum spanning trees into a single spanning tree.

Either argue that the algorithm correctly computes a minimum spanning tree of G, or provide an example for which the algorithm fails.

This algorithm fails. Take the following simple graph:



Never mind how the algorithm first divides the graph, the MST in one subgraph (the one containing u) will be one edge of cost 1, and in the other subgraph, it will be one edge of cost 8. Adding another edge of cost 1 will give a tree of cost 10. But clearly, it would be better to connect everyone to u via edges of cost 1.

3) 15 pts

Suppose you are given two sets A and B, each containing n positive integers. You can choose to reorder each set however you like. After reordering, let a_i be the ith element of set A, and let b_i be the ith element of set B. You then receive a payoff of $\prod_{i=1}^{n} a_i^{b_i}$. Give an algorithm that will maximize your payoff. Prove that your algorithm maximizes the payoff, and state its running time.

Solution:

Algorithm:
Sort A and B into monotonically decreasing order.

Proof:
Consider any indices i, j, p, q such that $i < j$ and $p < q$, and consider the terms ai^{bp} and aj^{bq}. We want to show that it is no worse to include these terms in the payoff than to include ai^{bq} and aj^{bp}, that is, $ai^{bp}\ aj^{bq} \geq ai^{bq}\ aj^{bp}$.

Since A and B are sorted into monotonically decreasing order and $i < j$, $p < q$, we have $ai \geq aj$ and $bp \geq bq$. Since ai and aj are positive and $bp - bq$ is nonnegative, we have $ai^{bp-bq} \geq aj^{bp-bq}$.

Multiplying both sides by $ai^{bq}\ aj^{bq}$ yields $ai^{bp}\ aj^{bq} \geq ai^{bq}\ aj^{bp}$.

Since the order of multiplication does not matter, sorting A and B into monotonically increasing order works as well.

Complexity:
Sorting 2 sets of n positive integers is O(nlogn).

4) 15 pts

Indicate, for each pair of expressions (A, B) in the table below, whether A is O, Ω or Θ of B. Assume that $k \geq 1$, and $\varepsilon > 0$ are constants. Answer "yes" or "no" in each box in the following form. No explanations required.

A	B	O	Ω	Θ
$\log^k n$	n^ε	Yes	No	No
$\sqrt{n}$	$n^{\sin n}$	No	No	No
$n^{\log m}$	$m^{\log n}$	Yes	Yes	Yes

The explanation is not required:

(1) $\log(\log n)=O(\log n) \Rightarrow k\log(\log n)=O(\varepsilon\log n) \Rightarrow \log(\log^k n)=O(\log n^\varepsilon) \Rightarrow \log^k n=O(n^\varepsilon)$

(2) $-1<=\sin n <=1$, so we can not determine which one is larger;

(3) $n^{\log m} =\Theta(n^{\lg m})$, $m^{\log n} =\Theta(m^{\lg n})$, let $m=10^x$, and $n = 10^y$, then $n^{\lg m} = m^{\lg n} = 2^{xy}$.

5) 15 pts
Suppose you are given a number x and an array A with n entries, each being a distinct number. Also it is known that the sequence of values A[1];A[2]; ... ;A[n] is unimodal. In other words for some unknown index p between 1 and n, we have
$A[1] < A[2] < ... < A[p]$ and $A[p] > A[p + 1] > ... > A[n]$.
Give an algorithm with running time O(log n) to find out if x belongs to A, if yes the algorithm should return the index j such that A[j] = x. You should justify both your algorithm and the running time.

Solution: The idea is to first find out p and then break A into two separated sorted arrays, then use binary search on these two arrays to check if x is belong to A.

Let FindPeak() be the function of finding the peak in A . Then FindPeak(A [1 : n]) works as follows:
Look at A [n / 2], there are 3 cases:
(1) If A [n / 2 - 1] < A [n / 2] < A [n/2+1], then the peak must come strictly after n / 2. Run FindPeak(A [n / 2 : n]).
(2) If A [n / 2 - 1] > A [n / 2] > A [n/2+1], then the peak must come strictly before n / 2. Run FindPeak(A [1 : n / 2]).
(3) If A [n / 2 - 1] < A [n / 2] > A [n/2+1], then the peak is A [n / 2], return n/2.

Now we know the peak index(p value). Then we can run binary search on A [1 : p] and A [p +1 : n] to see if x belong to them because they are both sorted.

In the procedure of finding p, we halve the size of the array in each recurrence. The running time T(n) satisfies T (n) = T (n/2) + O(1) . Thus T (n) = O (log n). Also both binary search has running time at most O (log n), so total running time is O (log n).

Explanations:
Note that trying to get x in one shot (in one divide and conquer recursion) usually does not work. Binary search certainly cannot work because the array is not sorted. Modified binary search can hardly work either. The problem is that in each round you need to abandon half the array. However, this decision is hard to made. For example, the array is [1 2 3 4 5 -1], x=-1. When divide we have mid=3. We find A[mid-1]<A[mid]<A[mid+1]. A common but wrong practice here is to continue search only in the A[1:mid]. We can see clearly x=-1 is in the second half of the array. On the other hand you cannot throw away the first half of the array either. The counter example is [1 2 3 4 5 -1] where x=1.

One other mistake is to search p in a linear fashion. This take O(n) in the worst case.

6) 20 pts
Given a string *S* with *m* digits (digits are from 1 to 9), you are required to delete *n* (*n*<*m*) digits from *S*. After the operation, you have a string *S*' with *m*-*n* digits, let *N* denote the number that *S*' represents. Design a greedy algorithm to minimize *N*. For example, *S*="456298111", *n* = 3, after deleting 4, 5 and 6, you get the minimal N=298111. Prove the correctness of your algorithm and analyze its time complexity.

Solution:

The greedy strategy: every step we delete a digit, make sure the number represented by the rest of the digits is minimal. To achieve this, in each step, we do the following operation:
Find the first *i* such that S[*i*]>S[*i*+1], then we delete S[i]; if such *i* does not exist, which means the digits are in increasing order, then we simply delete the last digit. (*)
We call the position i "peak".

Proof by induction
(1) We first prove (*) can achieve minimal *N* when n = 1. We have
$S' = a_1a_2\ldots a_{i-1}a_{i+1}\ldots a_{n-1}$
Without loss of generality, suppose we delete S[j] rather than S[i], the we have
$S_1 = a_1a_2\ldots a_{j-1}a_{j+1}\ldots a_{n-1}$ (j!=i), and N_1 is the number represented by S_1.
If $j < i$, since $a_j < a_{j+1}$, we have $N < N_1$; if $j > i$, since $a_{i+1} < a_i$, we have $N < N_1$.
Therefore, $N < N_1$ for all j!=i.
(2) Next, we assume (*) can achieve minimal N when n=k, now we prove (*) can achieve minimal N when n=k+1.
Suppose the first deletion, we delete p', then for the rest of k deletions, we need to do (*) k times (deleting the first k peaks) in order to get the minimal result, which is denoted by N';
Let N denote the number generated by deleting the first k+1 peaks, p1, p2, …, p_{k+1}, in S.
If p'=pi (1<=i<=k+1), then N'=N; if p'<p1 or p_i<p'<p_{i+1} or p'>p_{k+1}, similar to (1), we can prove N' > N; Therefore, we show that N<=N'.
(3) With (1) and (2), we prove the correctness of the algorithm.

Complexity:
O(n), but O(mn) is also acceptable.