CS 380 HW 1

C2C Caleb Song

Documentation: Got EI from Dr. Hadfield who helped me optimize some of my programs.

Question 1:

//Question 1: Write an algorithm that finds both the smallest and largest numbers in a list of n  
//numbers. Try to find a method that does at most 1.5n comparisons of array items.  
public static void findValues(int[] numArray, int length){  
 int min = numArray[0];  
 int max = numArray[0];  
 //Differentiate between odd and even when comparing with 2 values at a time  
 //Odd start for loop at 1 to compare two values at a time  
 // compare two values compare bigger to max and smaller to min  
 if(length%2 == 1){  
 for (int i = 1; i < length; i+=2) {  
 //see if first number is bigger than second number  
 if (numArray[i] > numArray[i + 1]) {  
 //if first number is larger then compare to max  
 if (numArray[i] > max) {  
 max = numArray[i];  
 }  
 //if first number is larger then compare second to min  
 if (numArray[i + 1] < min) {  
 min = numArray[i + 1];  
 }  
 } else {  
 //else see if second number is greater than max  
 if (numArray[i + 1] > max) {  
 max = numArray[i + 1];  
 }  
 //and see if first number is less than min  
 if (numArray[i] < min) {  
 min = numArray[i];  
 }  
 }  
 }  
 }  
 //Even start at index 0s  
 if(length%2 == 0){  
 for (int i = 0; i < length; i+=2){  
 if (numArray[i] > numArray[i+1]){  
 //if first number is larger then compare to max  
 if(numArray[i] > max){  
 max = numArray[i];  
 }  
 //if first number is larger then compare second to min  
 if (numArray[i+1] < min){  
 min = numArray[i+1];  
 }  
 }  
 else{  
 //else see if second number is greater than max  
 if (numArray[i+1]>max){  
 max = numArray[i+1];  
 }  
 //and see if first number is less than min  
 if (numArray[i] < min){  
 min = numArray[i];  
 }  
 }  
 }  
 }  
 //Print out min and max values  
 System.*out*.println("Minimum value: " + min);  
 System.*out*.println("Maximum value: " + max);  
}

Question 2:

//Given an undirected graph G(V, E) where V is the set of vertices and E is the set of edges, write an  
//algorithm to determine if the vertex set W (a proper subset of V) is a clique. A clique is a set of  
//vertices in a graph such that there is an edge from each vertex in the clique to every other vertex in  
//the clique. You may assume G is represented with an adjacency matrix and W is a simple 1D array  
//of vertex numbers  
public static boolean findClique(int[][] adjMatrix, int[] vertexSet){  
 //go through the vertex set and make sure that they are all in the adj matrix  
 for (int i = 0; i < vertexSet.length; i++){  
 for (int j = i+1; j < vertexSet.length; j++){  
 if(adjMatrix[vertexSet[i]][vertexSet[j]] != 1){  
 return false;  
 }  
 }  
 }  
 return true;  
}

Question 3:

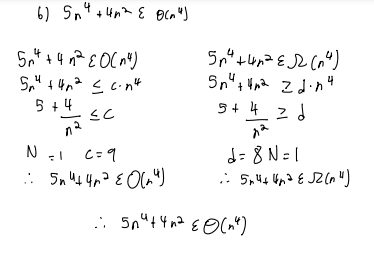
//Question 3: Write a Θ(n) algorithm that sorts n distinct integers, ranging in size between 1  
 //and kn inclusive, where k is a constant positive integer. (Hint: Use a knelement  
 //array.)  
 // Assume no duplicates in list  
 public static void sort(int[] unsorted, int k){  
 int n = unsorted.length;  
 int[] sortingArray = new int [k\*n+1];  
 int[] sorted = new int [n];  
 //Insert a 1 into sortingArray at an index if it appears as a number in unsorted array  
 for(int i = 0; i < n; i++){  
 sortingArray[unsorted[i]] = 1;  
 }  
 int counter = 0;  
 //Go through sortingArray and if an index is 1 insert it into sorted Array  
 for(int i = 1; i < sortingArray.length; i ++){  
 if (sortingArray[i] == 1){  
 sorted[counter] = i;  
 counter+=1;  
 }  
 }  
 //print out sorted array  
 for(int i = 0; i < n; i++){  
 System.*out*.print(sorted[i] +" ");  
 }  
  
 }  
}

Question 4: Algorithm A performs 10n^2 basic operations, and algorithm B performs 300 lnn basic operations. For what value of n does algorithm B start to show its better performance?

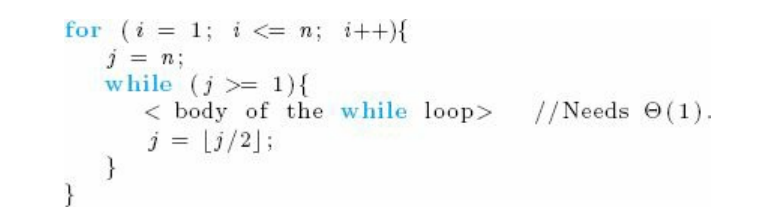
 n = 8

Question 5:

Question 6:

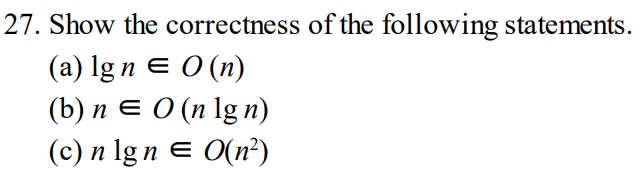


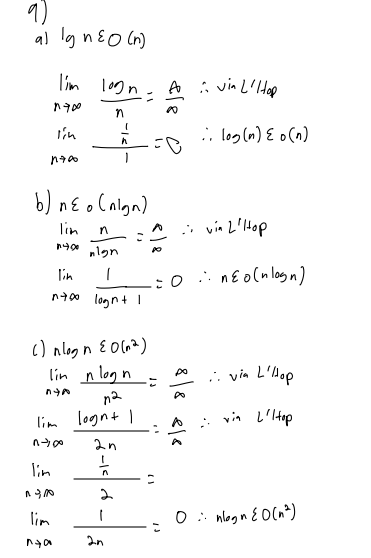
Question 7:



Question 8:

Question 9





Question 10

