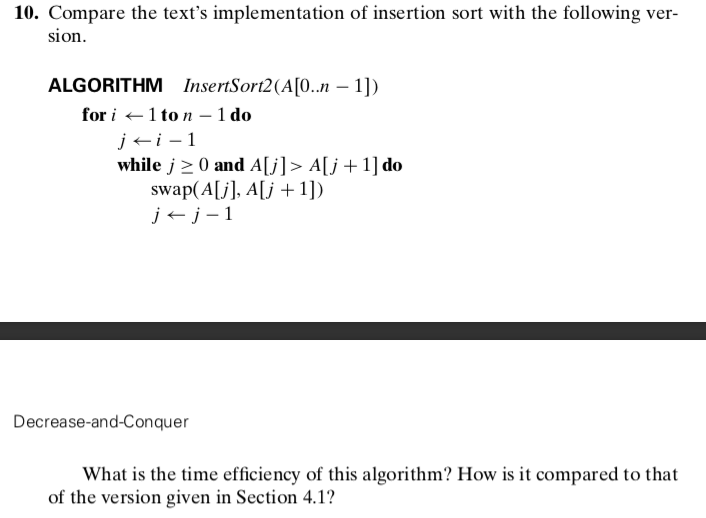
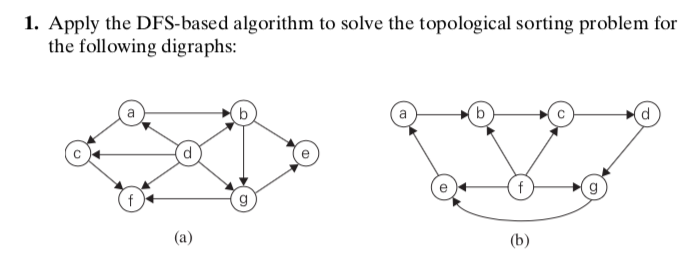
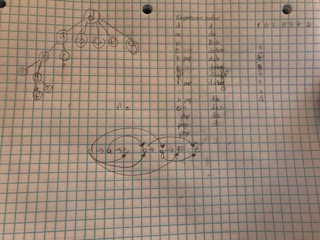
CS4245: Assignment 2

* + 1. *A stack of fake coins* **There are n stacks of n identical-looking coins. All of the coins in one of these stacks are counterfeit, while all the coins in the other stacks are genuine. Every genuine coin weighs 10 grams; every fake weighs 11 grams. You have an analytical scale that can determine the exact weight of any number of coins.**
  1. **Devise a brute-force algorithm to identify the stack with the fake coins and determine its worst-case efficiency class.**
* Divide and conquer:
  + Begin:
  + arStacks[n] // stores all stacks of coins in each element
  + weightStack[n] // stores the weight
  + for i < n
    - if weightStack[i] < weightStack [i+1]
      * return arStack[i+1] // i+1 is heavier so it contains the fake
    - if weightStack[i] > weightStack[i+1]
      * return arStack[i] //I is heavier so it has the fakces
    - // if equal they are both real continue shifitng
  + return None
* worst case scenario will be O(n) if the last element of the array contained the counterfeit
  1. **What is the minimum number of weighings needed to identify the stack with the fake coins?**
* The minimum number of weighs are based on the number of stacks so if there are a n amount of stacks then their will be n amount of weighing.

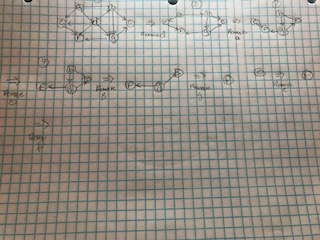
4.1.10 

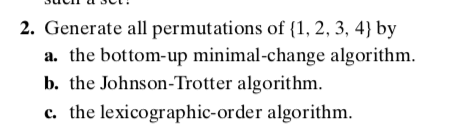
* The best case is when A[j] is less then A[j+1] on every iteration, so its basically when it is already sorted. So the for loop will keep running and the while will only do the comparison, making the most important operation being the comparison of A[j] >A[j+1]. This will be executed (n-1), making it Θ(n)
* For the worst case of this, the for loop will execute n-1 and the inner while loop will execute true for every iteration. This makes the worst the case (n)(n-10)/2 == Θ(n2).
* The average case will be in between the best and worst making it (n2)/4 == Θ(n2)
* Based on the texts and the analysis of this algorithm the efficiencies are the same.

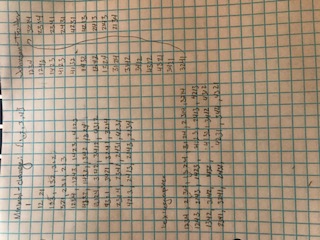
4.2.1a

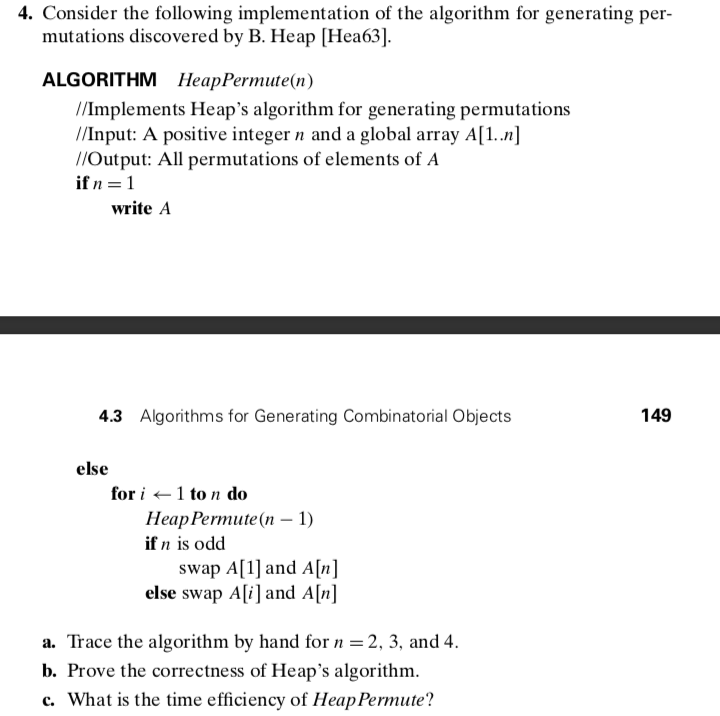
* 

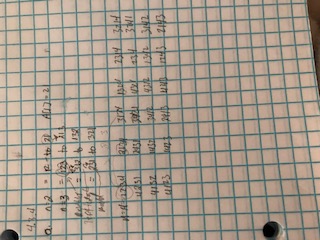
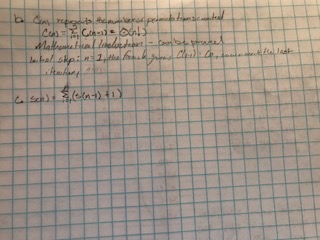
4.2.5a Apply the source-removal algorithm to the diagraphs of Problem 1 above:



4.3.2 



4.3.4 

1. a
2. C(n) is the number of times a new permutation is created. 

Write a program that utilizes linear interpolation to find an element for a sorted list of numbers. Compare the linear interpolation implementation to a binary search implementation (you may use a source on the internet for the binary search algorithm, but be sure to cite the source). Use the datasets provided for the analysis (there are three datasets in the file). Be sure to consider both when the element is in the list and when it is not present in the list.

Please review additional documents

Comparison values are:

Interpolation:

Total Time:

0.00122380256653

Average Time:

2.78136946938e-05

BST:

Total Time:

0.020653963089

Average Time:

0.000469408252022

BST was atrociously slow, whereas Interpolation was extremely quick and efficient. The BST wasn’t able to hold all of the nodes required to build the full tree so I had to reduce the amount of input.