CSC 310 Homework

Chapter 4, Part 1

4.1: 2a, 3 ,10

4.2: 1a-b, 5, 9

4.3: 1, 2a-c, 3, 4

* 4.1.2a
  + Assume that all the glasses are numbered from 1 to 2n, left to right. Pour the soda from glass 2 into the second to last glass (2n-1). This creates an alternating pattern between the first and last pair of glasses. Therefore reducing the original problem to 2(n-2) glasses. If n is even this alternation will need to be done floor(n/2) times. If the glasses are odd, this alternation will need to be done floor((n-1)/2). Since any algorithm must move at least one filled glass, floor(n/2) is the least number of moves needed to solve the problem.
* 4.1.3
  + If n=2, the solution must be (below) as each cell has an odd number of cells next to it. Connections are shown in parentheses

|  |  |  |  |
| --- | --- | --- | --- |
|  | x(1) | x(1) |  |
|  |  |  |  |

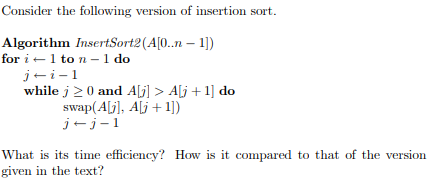
* + Therefore, marking two cells adjacent to the left most cell provides a solution to n=4

|  |  |  |  |
| --- | --- | --- | --- |
|  | x(1) |  |  |
| x(1) | x(3) | x(1) |  |
|  |  |  |  |

* + Following the same pattern for n=6, the two cells adjacent to the right most cell provide a solution

|  |  |  |  |
| --- | --- | --- | --- |
|  | x(1) |  |  |
| x(1) | x(3) | x(3) | x(1) |
|  |  | x(1) |  |
|  |  |  |  |

* + By following this same pattern for all even values of n, we can provide a solution for all even values of n.
* 4.1.10



The efficiency class will be the same as insertion sort. The innermost loop of InsertSort2 contains a swap (3 assignments) and a decrement while the other insertion sort only has 1 assignment and 1 decrement. Ignoring the time on the decrement the running time ratio would be

* 4.2.1
  + a)
    - A -> BC
    - B -> GE
    - C -> F
    - D -> ABCFG
    - E -> none
    - F -> none
    - G -> FE
    - The vertices are popped of the stack as follows -> efgbcad
    - The Topological order from reversing the list is -> dacbgfe
  + b)
    - A->B
    - B->C
    - C->D
    - D->G
    - E->A
    - F->EBCG
    - G->E
    - The DFS traversal starts at a and encounters an edge from e that returns it back to a making the list -> egdcba
* 4.2.5
  + By applying the same source-removal algorithm to the digraphs to problem 1 we get the following:
  + a)
    - Remove d, remove a, remove b, remove, c, remove g, remove e, remove f, making the topological ordering -> dabcgef
  + b)
    - Topological sorting is impossible on the second digraph since the only node that can be removed is f. Leaving the traversal A->B->C->D->G->E->A.
* 4.2.9
  + a)
    - First dfs:

f1 g2 b3 a4 d5 h6 e7 c8

* + - Edge reversal
    - Seconds dfs:

c->e->h->end d->end a->f->g->b->end

* + - Strongly connected components are (a, b, f, g), (d), and (c, h, e)
  + b)
    - For adjacency matrix of size n \* n, where n is number of vectors:

first dfs = Θ(n2)

Edge reversal = Θ(n2)

Seconds dfs = Θ(n2)

Total efficiency = Θ(n2) + Θ(n2) + Θ(n2) = 3 Θ(n2) ∈ Θ(n2)

* + - For list representation of size n + e, where e is number of edges:

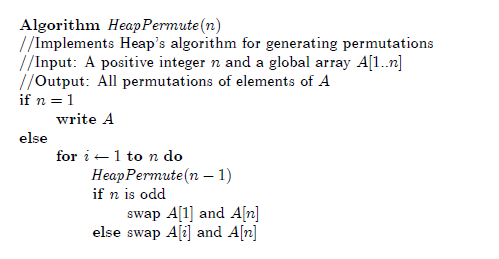
first dfs = Θ(n + e)

edge reversal = Θ(e)

Second dfs = Θ(n + e)

Total efficiency = Θ(n+e) + Θ(e) + Θ(n+e) = 2 Θ(n+e) + Θ(e) ∈ Θ(n + e)

* + c)
    - a directed acyclic graph has zero strongly connected components
* 4.3.1
  + It is not realistic for an algorithm to generate all the subsets of a 25-element set as 25! is too large, but generating the subsets, 225,would be much more manageable
* 4.3.2
  + a)
    - 1
    - 12 21
    - 123 132 312
    - 321 231 213
    - 1234 1243 1423 4123
    - 4132 1432 1342 1324
    - 3124 3142 3412 4312
    - 4321 3421 3241 3214
    - 2314 2341 2431 4231
    - 4213 2413 2143 2134
  + b)
    - 1234 1243 1423 4123
    - 4132 1432 1342 1324
    - 3124 3142 3412 4312
    - 4321 3421 3241 3214
    - 2314 2341 2431 4231
    - 4213 2413 2143 2134
  + c)
    - 1234 1243 1324 1342 1423 1432
    - 2134 2143 2314 2341 2413 2431
    - 3124 3142 3214 3241 3412 3421
    - 4123 4132 4213 4231 4312 4321
* 4.3.3
  + 1223, 1232, 1322, 2123, 2132, 2213, 2231, 2312, 2321, 3122, 3212, 3221
  + yes the permutations are generated correctly.
* 4.3.4



* + a)
    - n=2
      * 12 21
    - n=3
      * 123 213
      * 312 132
      * 231 321
    - n=4
      * 1234 2134 3124 1324 2314 3214
      * 4231 2431 3421 4321 2341 2341
      * 4132 1432 3412 4312 1342 3142
      * 4123 1423 2413 4213 1243 2143
  + b)
    - C(n) = n C(n-1) = n!, because n! is equivalent to the number of permutations that a list has it can be confirm that all permutations are generated
  + c)
    - The number of swaps can be represented in the form S(n) = nS(n-1)+n, then we can divide both sides by n! Or (n-1)!\*n: . T(n) = s(n)/n! so, T(n) = T(n-1) + 1/(n-1)!. Though backward substitution T(n) = . With this we can do S(n) = n!, with is in the growth order of