CSC 3110
Homework 2
May Wandyez
Gq5426
2/12/2024

CSC 3110

Algorithm Design and Analysis

(30 points)

Due: 02/14/2024 11:59 PM

Note: Submit answers in PDF document format. Please read the submission format for appropriate file naming conventions.

Note to Professor:

Professor, have you noticed that the sum total of the problems assigned amounts to 5 total pages of problems? This is composed of 13 different problems containing 25 different parts. It is one thing to look at these problems by just referring them by their exercise numbers while assigning, it is another entirely to have them all out in front of you.

Think: how long would it take on average to complete each part for an average student? Assuming the student is studious, and takes the time to research the problem to ensure that they got the correct answer, let's say they take around 9 minutes per part. This 9 minutes would entail recognizing the core concept of the problem (1 minute), looking it up in the textbook or notes or online (2 minutes), reading the concept(3 minutes), and applying it correctly on the first try (3 minutes). This would mean that an average student would spend around 225 minutes to complete this homework assignment. Averaged out per week this means that this assignment would take on average 112 minutes per week.

Perhaps a way to better see the scale when assigning problems would be to write down the problems themselves instead of just the numbering of the exercises? This may give a better insight into the scale of the problems being assigned.

This homework assignment takes longer during the week than all current assignments across 4 different higher level programming courses combined. For reference my current programming courses are:

CSC 3100 – Computer Architecture and Organization (4 credits)
-Assignments: Weekly homework/labs(~30 minutes per week)

CSC 4110 – Software Engineering(4 credits)
-Assignments: Monthly Projects(~30 minutes per week), Weekly homeworks(~30 minutes

CSC 3110
Homework 2
May Wandyez
Gq5426
2/12/2024

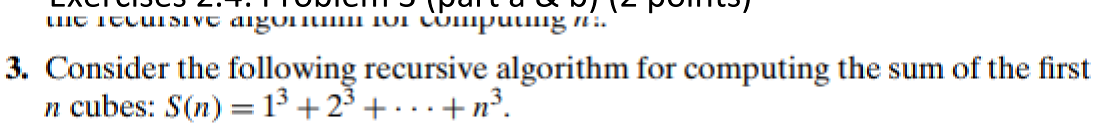
per week)

CSC 4710 – Introduction: Database Management Systems (3 Credits)
-Assignments: bi-weekly assignments(~15 minutes per week, averaged)

CSC 4760 – Introduction To Deep Learning (3 credits)
-Assignments: Weekly homework (~30 minutes per week)

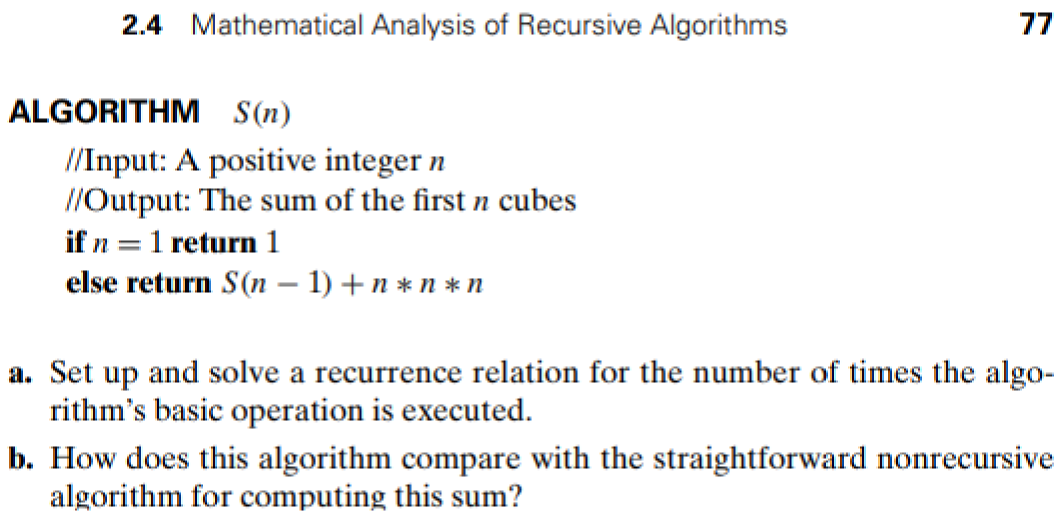
CSC 3110 – Algorithm Design and Analysis (3 credits)
-Assignments: bi-weekly homework(~112 minutes per week)

1) Exercises 2.4: Problem 3 (part a & b) (2 points)



the recursive algorithm for computing $n!$.

3. Consider the following recursive algorithm for computing the sum of the first n cubes: $S(n) = 1^3 + 2^3 + \cdots + n^3$.



2.4 Mathematical Analysis of Recursive Algorithms 77

```
ALGORITHM S(n)
    //Input: A positive integer n
    //Output: The sum of the first n cubes
    if n = 1 return 1
    else return S(n − 1) + n ∗ n ∗ n
```

a. Set up and solve a recurrence relation for the number of times the algorithm's basic operation is executed.

b. How does this algorithm compare with the straightforward nonrecursive algorithm for computing this sum?

a.) The basic operation is *, which is exercised twice on every pass.
You can set up the number of executions of the basic operation as
f(n) = f(n-1) + 2*1
Where n(1) =0 as zero executions of the basic operation are committed on the last pass.

We therefore know that f(n=1) = 0
Therefore we can create a substitution, where we set n-1 = 1

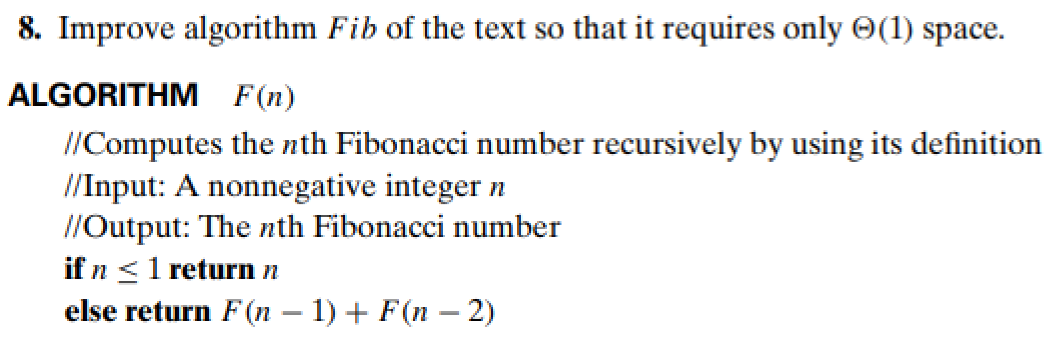
The equations therefore becomes:
f(n) = f(1) +2*(n-1)
f(n) = 0 + 2*(n-1)
f(n) = 2*(n-1)

b.) We have to write out the non recursive way of writing this sum:

```
sum = 0
i = 1
for I in range (2,n):
    sum += i*i*i
return i
```

The basic operation is * and it is executed 2 times for every pass, for n-1 passes, which means the efficiency of the nonrecursive implementation is O(2*(n-1)) which is exactly the same as the answer in part a.

2) Exercises 2.5: Problem 8 (2 points)


8. Improve algorithm *Fib* of the text so that it requires only $\Theta(1)$ space.

```
ALGORITHM  F(n)
    //Computes the nth Fibonacci number recursively by using its definition
    //Input: A nonnegative integer n
    //Output: The nth Fibonacci number
    if $n \leq 1$ return n
    else return $F(n-1) + F(n-2)$
```

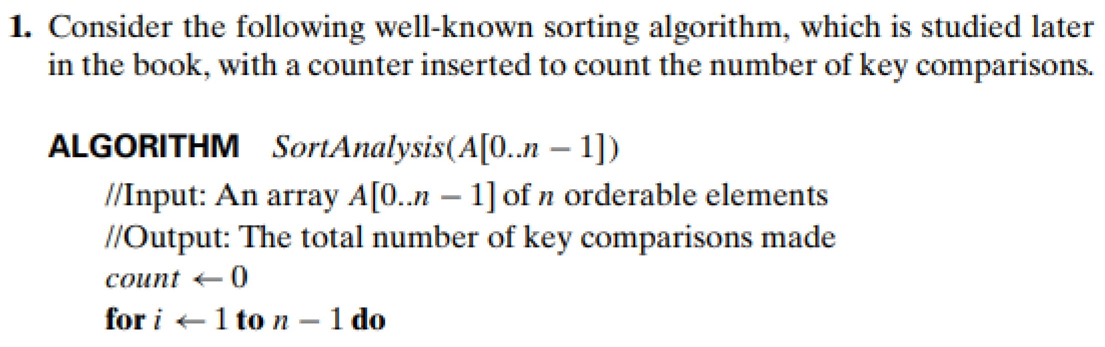
This question is asking to produce a Fibonacci algorithm with constant (theta(1)) time efficiency. To be clear, to accurately do so is not possible, it always involves some form of rounding that is subject to computer errors. Here is an implementation of that using Binet's formula. Binet's formula calculates the nth Fibonacci number, we can slap this into the code so we don't have to do more than one calculation. The only issue is that you have to use a built in rounding function in your algorithm -this isn't hard – but the inaccuracy is the price you pay for constant time implementation:

```
F(n)
return round( (1/sqrt(5)) *  (    ((1+sqrt(5))/2)^n  -  ((1-sqrt(5))/2)^n   )
```

CSC 3110
Homework 2
May Wandyez
Gq5426
2/12/2024

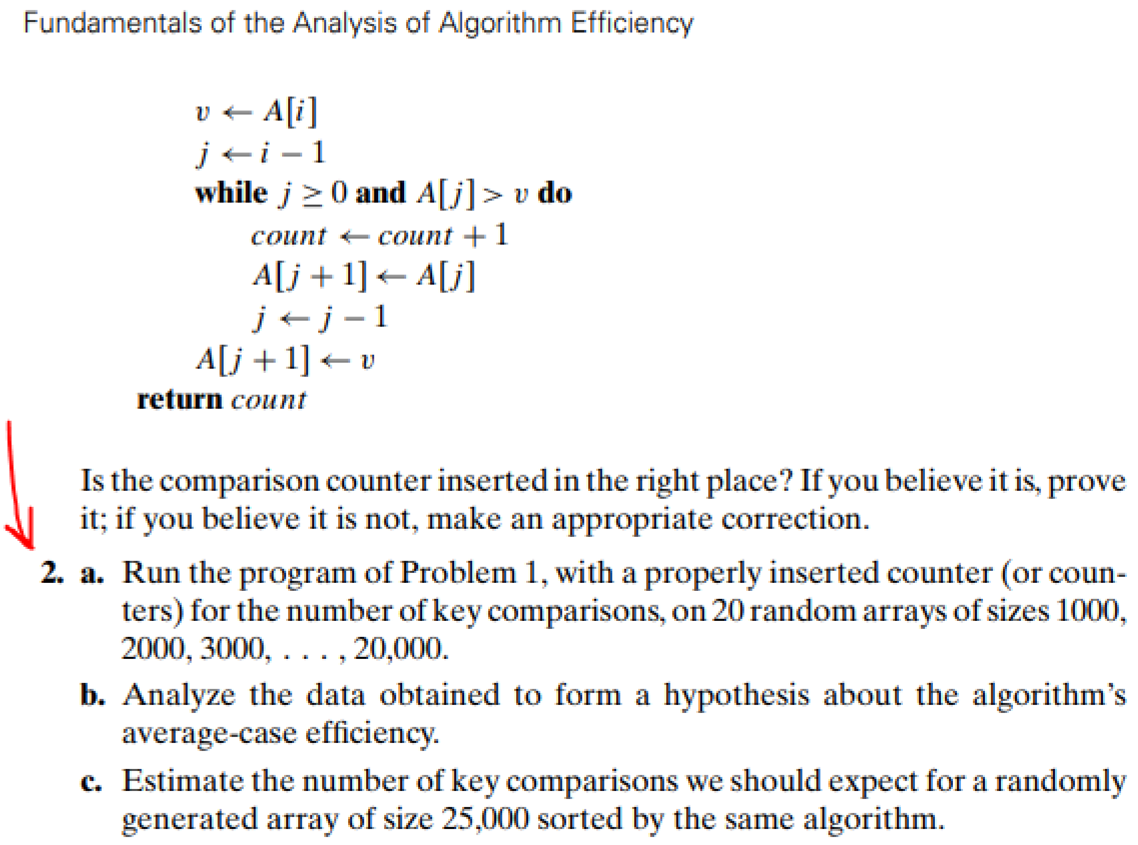
. Another good way to get constant time implementation is to pre-calculate a bunch of Fibonacci numbers and put them into an array and then access the nth member of that array.

3) Exercises 2.6: Problem 2 (parts a-c) (2 points)



1. Consider the following well-known sorting algorithm, which is studied later in the book, with a counter inserted to count the number of key comparisons.

```
ALGORITHM  SortAnalysis(A[0..n − 1])
    //Input: An array A[0..n − 1] of n orderable elements
    //Output: The total number of key comparisons made
    count ← 0
    for i ← 1 to n − 1 do
```



Fundamentals of the Analysis of Algorithm Efficiency

```
        v ← A[i]
        j ← i − 1
        while j ≥ 0 and A[j] > v do
            count ← count + 1
            A[j + 1] ← A[j]
            j ← j − 1
        A[j + 1] ← v
    return count
```

Is the comparison counter inserted in the right place? If you believe it is, prove it; if you believe it is not, make an appropriate correction.

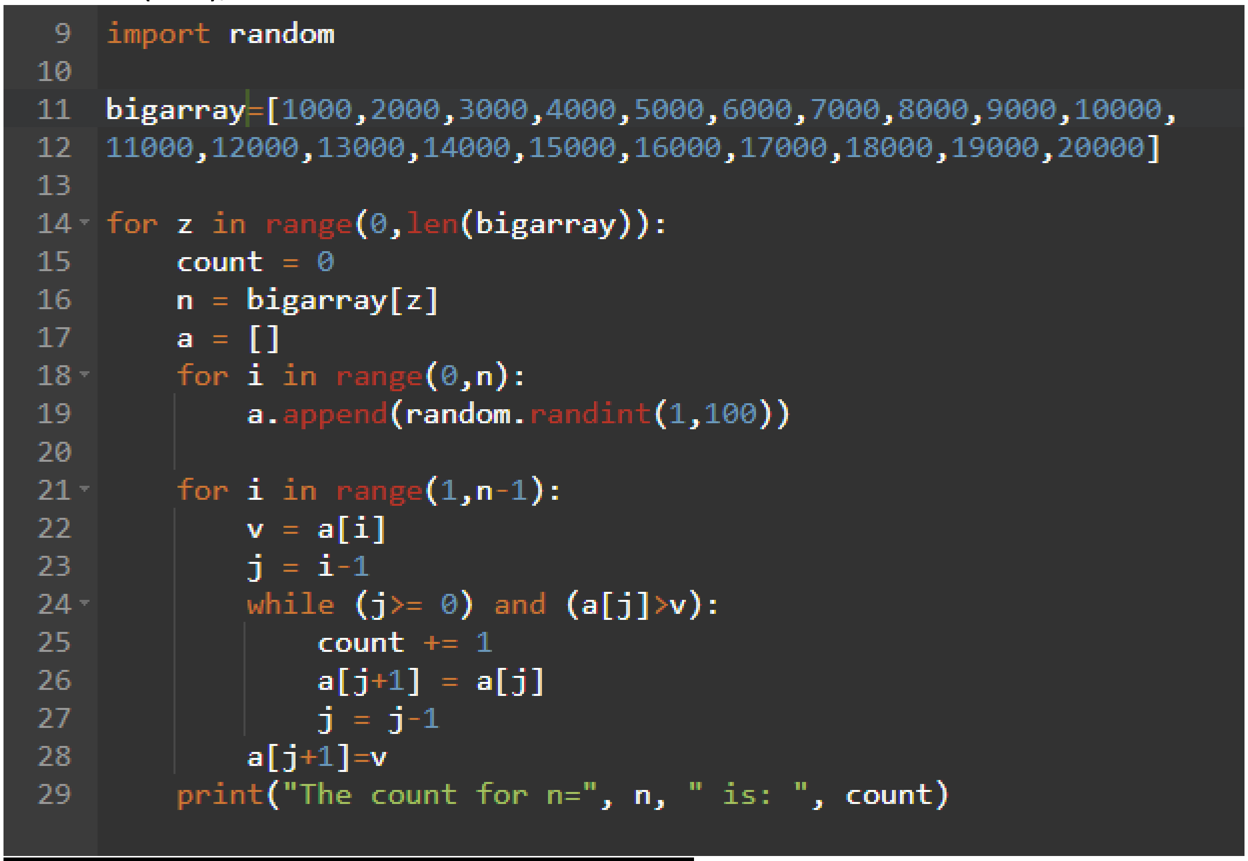
2. a. Run the program of Problem 1, with a properly inserted counter (or counters) for the number of key comparisons, on 20 random arrays of sizes 1000, 2000, 3000, . . . , 20,000.

b. Analyze the data obtained to form a hypothesis about the algorithm's average-case efficiency.

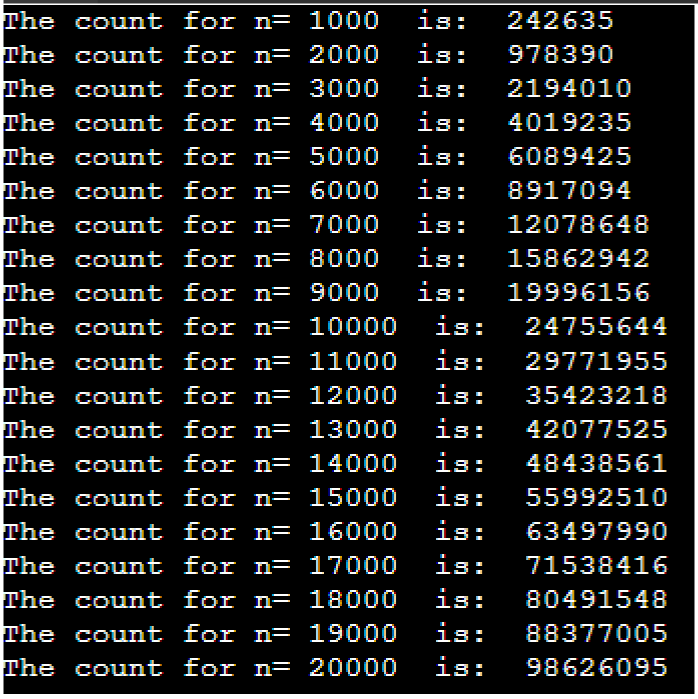
c. Estimate the number of key comparisons we should expect for a randomly generated array of size 25,000 sorted by the same algorithm.

CSC 3110
Homework 2
May Wandyez
Gq5426
2/12/2024

a.) The above algorithm is a while loop within a for loop, with the n loop running (n-1) times and the while loop running from j times. This roughly looks like a n^2 efficiency case. However J’s size decreases with every pass. Actual results are as follows ~(n^2)/4:



```python
import random

bigarray=[1000,2000,3000,4000,5000,6000,7000,8000,9000,10000,
11000,12000,13000,14000,15000,16000,17000,18000,19000,20000]

for z in range(0,len(bigarray)):
    count = 0
    n = bigarray[z]
    a = []
    for i in range(0,n):
        a.append(random.randint(1,100))

    for i in range(1,n-1):
        v = a[i]
        j = i-1
        while (j>= 0) and (a[j]>v):
            count += 1
            a[j+1] = a[j]
            j = j-1
        a[j+1]=v
    print("The count for n=", n, " is: ", count)
```



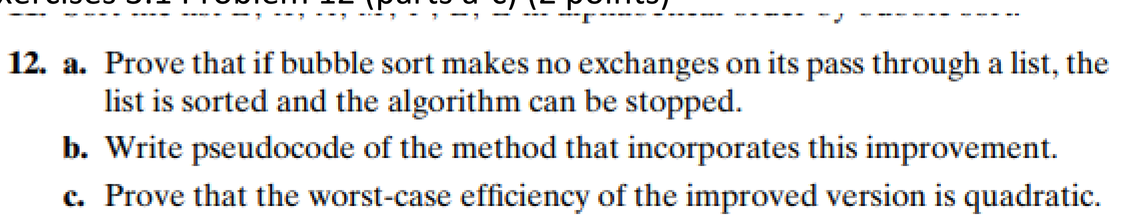
```
The count for n= 1000  is:  242635
The count for n= 2000  is:  978390
The count for n= 3000  is:  2194010
The count for n= 4000  is:  4019235
The count for n= 5000  is:  6089425
The count for n= 6000  is:  8917094
The count for n= 7000  is:  12078648
The count for n= 8000  is:  15862942
The count for n= 9000  is:  19996156
The count for n= 10000  is:  24755644
The count for n= 11000  is:  29771955
The count for n= 12000  is:  35423218
The count for n= 13000  is:  42077525
The count for n= 14000  is:  48438561
The count for n= 15000  is:  55992510
The count for n= 16000  is:  63497990
The count for n= 17000  is:  71538416
The count for n= 18000  is:  80491548
The count for n= 19000  is:  88377005
The count for n= 20000  is:  98626095
```

b.)The efficiency of the algorithm’s average-case efficiency is theta(n^2) type.

c.) Given that my random array sorter got around n^2/4 for the number of key comparison made, we would expect similar results(10,000^2)/4 = 25000000

CSC 3110
Homework 2
May Wandyez
Gq5426
2/12/2024

4) Exercises 3.1 Problem 12 (parts a-c) (2 points)


12. a. Prove that if bubble sort makes no exchanges on its pass through a list, the list is sorted and the algorithm can be stopped.

b. Write pseudocode of the method that incorporates this improvement.

c. Prove that the worst-case efficiency of the improved version is quadratic.

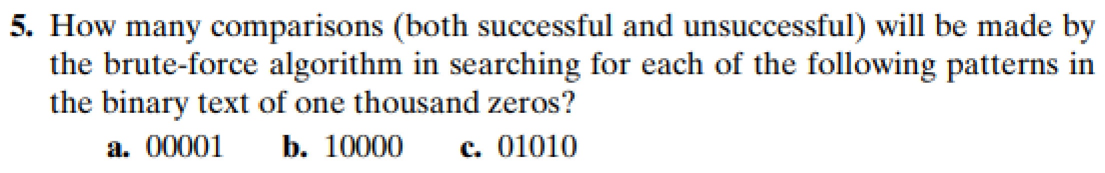
a.) A bubble sort operates by swapping elements if the next element is larger than the current element. If no swaps are made, it means that EACH element is larger than the next, which means that the list is sorted. Because the list is sorted this means that a bubble sort can be stopped when it determines that a pass has resulted in no swaps.

b.)

```
For array A size n
For z in range(0,n):
        For I in range(0,n-z):
                Swap = false
                If A[i+1] < A[j]:
                        Swap(A[i],A[i+1])
                        Swap = true
        If swap == false:
                Break:
```

c.) The worst case efficiency means that array is organized in decreasing order. This means that each element is smaller than the next, which means that a swap will always occur. Because the improvement never comes into effect, it means that the improvement has the same worst case efficiency as the original version: quadratic (n^2)

5) Exercises 3.2 Problem 5 (parts a-c) (2 points)


5. How many comparisons (both successful and unsuccessful) will be made by the brute-force algorithm in searching for each of the following patterns in the binary text of one thousand zeros?

a. 00001 **b.** 10000 **c.** 01010

Let’s talk about the brute force algorithm first. It begins with element being compared, then compares the next subsequent elements to ensure that they match. Let n be the number digits of the binary number being searched through, and l represent the number of digits binary number being searched for.

For efficiency, we can say there are n-l-1 numbers to comb through (we can’t

CSC 3110
Homework 2
May Wandyez
Gq5426
2/12/2024

keep comparing when the length of the number to search for is greater than the remaining number of digits to search through), and each comparison must make l comparisons. This would mean there are (n-l-1)*l comparisons to make at maximum

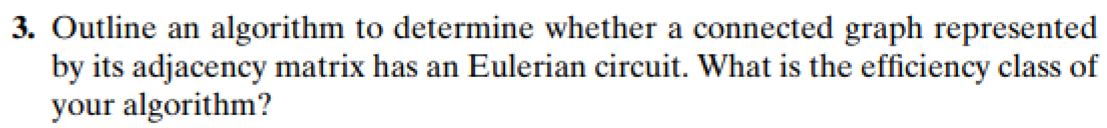
BUT WAIT, n is composed entirely of zeroes, which means that the number of comparisons changes to a different number which is the first digit that represents a non zero(let's call it NZ for first non-zero). This means the new equation would be (n-l-1)*NZ.

A.) N = 1000, l = 5, NZ =5
(1000-5-1)*5 = 4980 comparisons

B.) N= 1000, l =5, NZ =1
(1000-5-1)*1 = 996 comparisons

C.) N =1000, l = 5, NZ =2
(1000-5-1)*2 = 1992 comparisons

6) Exercises 3.4 Problem 3 (2 points)


3. Outline an algorithm to determine whether a connected graph represented by its adjacency matrix has an Eulerian circuit. What is the efficiency class of your algorithm?

A eulerian circuit is a traversal of a graph that uses every edge EXACTLY ONCE AND NOT MORE THAN ONCE. If a vertex has an odd number of edges, it cannot be traversed successfully without backtracking because the path has to come and go an equal number of times from a vertex. This means what we are checking for is if any single vertex has an odd number of edges.

The algorithm would proceed as follows:
n represents a matrix of width n x n
v represents a vertex in the matrix

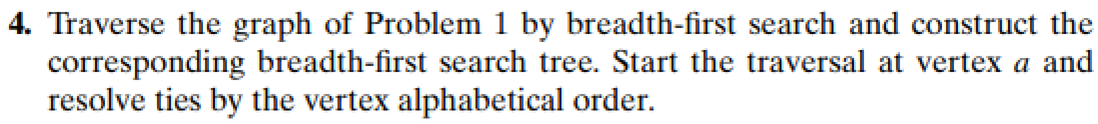
```
def eulerianTrueorFalse(n)
for j in range(0,n):
        for i in range (0,n):
                If v[j][i]%2 != 0
                        Return false
Return true
```

The efficiency of this algorithm is n^2, since there is only one comparison made for every vertex for a matrix of n x n.

CSC 3110
Homework 2
May Wandyez
Gq5426
2/12/2024

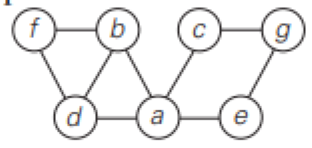
7) Exercises 3.5:

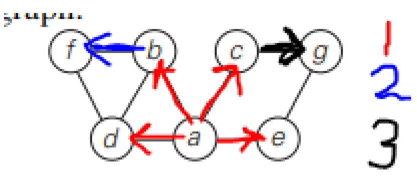
a. Problem 4 (2 points)



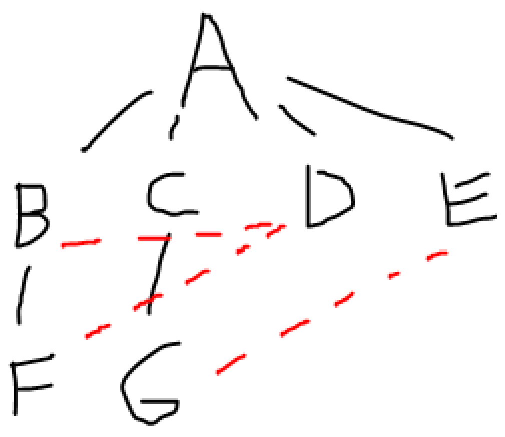
4. Traverse the graph of Problem 1 by breadth-first search and construct the corresponding breadth-first search tree. Start the traversal at vertex a and resolve ties by the vertex alphabetical order.

1. Consider the following graph.





Traversing the above graph leads to the array of [a,b,c,d,e,f,g] the resulting tree looks like this:

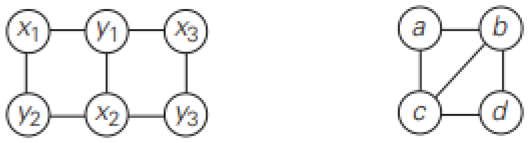


Wow what a tree. Look at that graphical quality.

b. Problem 8 (part a & b) (2 points)

CSC 3110
Homework 2
May Wandyez
Gq5426
2/12/2024

8. A graph is said to be ***bipartite*** if all its vertices can be partitioned into two disjoint subsets X and Y so that every edge connects a vertex in X with a vertex in Y. (One can also say that a graph is bipartite if its vertices can be colored in two colors so that every edge has its vertices colored in different colors; such graphs are also called ***2-colorable***.) For example, graph (i) is bipartite while graph (ii) is not.



(i) (ii)

a. Design a DFS-based algorithm for checking whether a graph is bipartite.
b. Design a BFS-based algorithm for checking whether a graph is bipartite.

So the problem here is determining if the vertices can be partitioned into subsets such that every vertex has 2 different colors. This would not be possible there was a edge connecting a vertex to a vertex on the same level. This means that a way to check if the graph can be partitioned is to visit a vertex, mark it with a color ‘RED’ , check all edges to determine if the same color is used and if so then the entire graph is not partitionable, if this check is passed then mark all edges leading from that vertex ‘BLUE’ and continue the search pattern.

A.) DFS (Sorry, the spacing got weird), the gist of this is that it compares the color of all edges against each other when it reaches a new edge, and if any colors mismatch then it breaks the loop).

```
    function DFSTorF(graph, root):
    stack=[start] //this is the stack, it tells us who is next to visit
visited = visited //this is where we have been, no point in going back.
red[start] //stack of rednodes
blue[] //list of bluenodes

while len(stack != 0):
    currentnode = stack.pop*()
    if currentnode is not in visited:
        visited.add(currentnode)
        //add the edges of currentnode to the stack, and //check to see if any
are identical

        For neighborA in graph[currentnode]:
```

CSC 3110
Homework 2
May Wandyez
Gq5426
2/12/2024

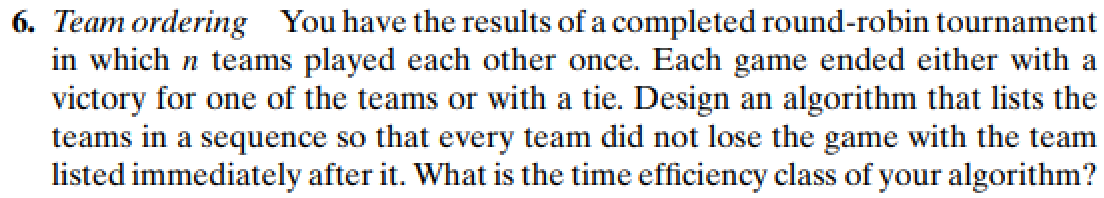
```
For neighborB in graph[currentnode]:
            if neighbor is not in visited:
                stack.push(neighbor)
                if currentnode== red:
                    blue.append(neighbor)
                else:
                    red.append(neighbor)
            elseif neighbor.getcolor() == currentnode.getcolor()
                return false

return true
```

B.) BFS – the gist of this is basically the same, while traversing the graph, check to see if any of the edges are mismatched in color, if so then return false.

```
    Function BFS(graph, start):
    Queue = start[]
    Visited = []
    Red = [start]
    Blue = []

    While(len(queue)!=0):
        Currentnode = queue.dequeue()
        If currentnode is not in visited
            Visited.add(currentnode)
            For neighbor in graph[currentnode]:
                If neighbor is not in visited:
                    Queue.enque(neighbor)
                    If currentnode.getcolor() == red:
                        Blue.append(currentneighbor)
                    Else:
                        Red.append(currentneighbor)
                elseif neighbor.getcolor() == currentnode.getcolor():
                    return false
    return true
```

8) Exercises 4.1 Problem 6 (2 points)

CSC 3110
Homework 2
May Wandyez
Gq5426
2/12/2024

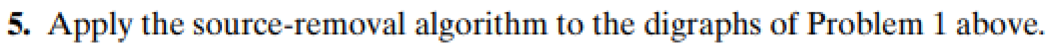


6. *Team ordering* You have the results of a completed round-robin tournament in which n teams played each other once. Each game ended either with a victory for one of the teams or with a tie. Design an algorithm that lists the teams in a sequence so that every team did not lose the game with the team listed immediately after it. What is the time efficiency class of your algorithm?

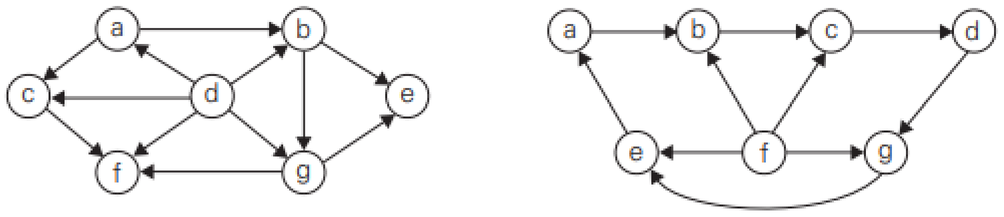
The objective here is to place every individual team next to a team it did not lose to. This means that for each team in the list, the entire list must be scanned and the team should be inserted in the list position before the first team it did not lose to. If the team lost to every team it should be placed at the end of the list so that there are no teams immediately after it. Because the list has to be passed through for each team, and then each team must be compared against each team, this is a loop within a loop, which suggest n^2 efficiency.

9) Exercises 4.2

a. Problem 5 (2 points)



5. Apply the source-removal algorithm to the digraphs of Problem 1 above.



(a) (b)

The source removal algorithm works by repeatedly identifying a vertex with no incoming edges, adding this verted to a sorted list, removing all edges attached to this vertex, then repeating until the list is empty. However, if all remaining edges point to each other, then topological sorting is impossible due to the presence of a cycle.

CSC 3110
Homework 2
May Wandyez
Gq5426
2/12/2024

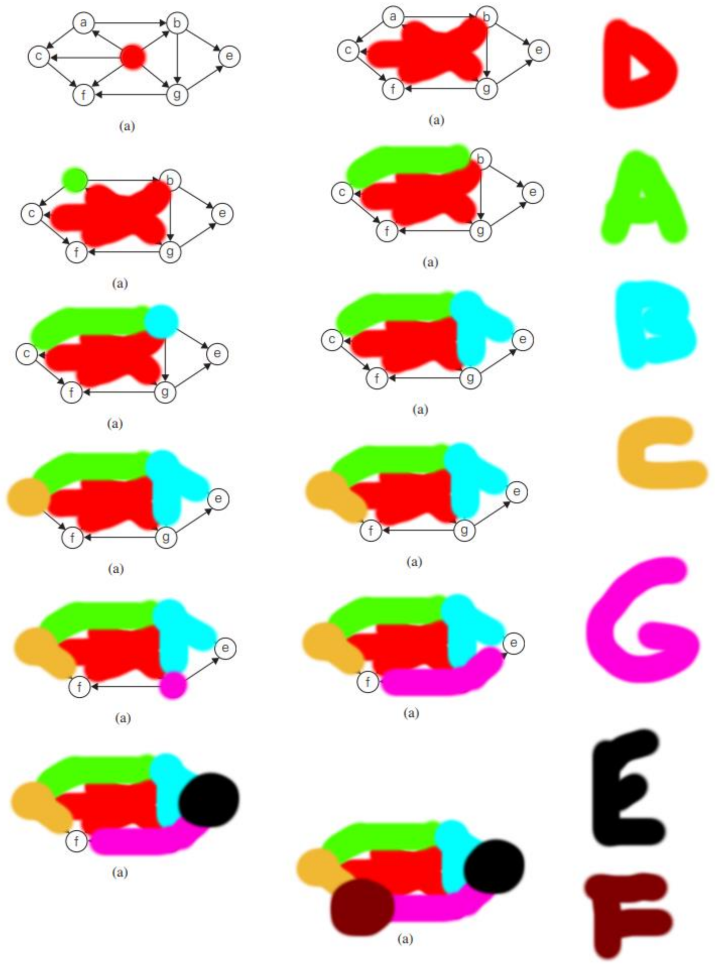


Figure 8-A: Obtained Order DABCGEF

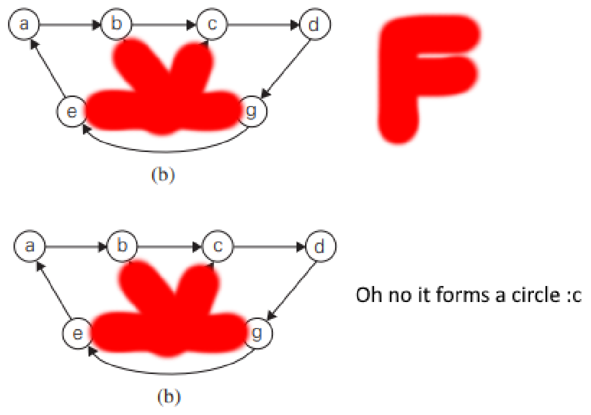


Figure 8-B Cycle Presence Means Topological Sorting Is Impossible

b. Problem 9 (parts a-c) (2 points)

CSC 3110
Homework 2
May Wandyez
Gq5426
2/12/2024

9. A digraph is called ***strongly connected*** if for any pair of two distinct vertices u and v there exists a directed path from u to v and a directed path from v to u. In general, a digraph's vertices can be partitioned into disjoint maximal subsets of vertices that are mutually accessible via directed paths; these subsets are called ***strongly connected components*** of the digraph. There are two DFS-

based algorithms for identifying strongly connected components. Here is the simpler (but somewhat less efficient) one of the two:

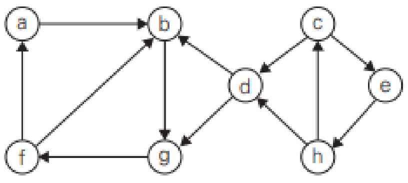
Step 1 Perform a DFS traversal of the digraph given and number its vertices in the order they become dead ends.

Step 2 Reverse the directions of all the edges of the digraph.

Step 3 Perform a DFS traversal of the new digraph by starting (and, if necessary, restarting) the traversal at the highest numbered vertex among still unvisited vertices.

The strongly connected components are exactly the vertices of the DFS trees obtained during the last traversal.

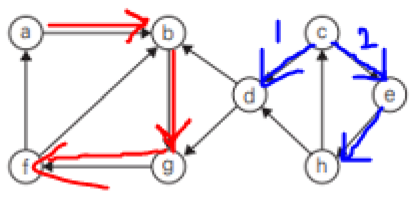
a. Apply this algorithm to the following digraph to determine its strongly connected components:



b. What is the time efficiency class of this algorithm? Give separate answers for the adjacency matrix representation and adjacency list representation of an input digraph.

c. How many strongly connected components does a dag have?

a.) Traversing the given digraph starting at a leads to the following traversal: a4,b3,g2,f1, c8,d5, e7,h6



Reversing the edges leads to:

CSC 3110
Homework 2
May Wandyez
Gq5426
2/12/2024

$$\begin{tikzcd}
a^4 \arrow[dd] & b^3 \arrow[l] \arrow[ddl] \arrow[dr] & & c^8 \arrow[dd] & \\
& & d^5 \arrow[ur] \arrow[dr] & & e^7 \arrow[ul] \\
f^1 \arrow[r] & g^2 \arrow[uu] \arrow[ur] & & h^6 \arrow[ur] &
\end{tikzcd}$$

This leads to the traversal of: [c,h,e],[d],[a,f,g,b] which means that these subgraphs are the strongly connected components of the original digraph.

b.) Adjacency Matrix:
An adjacency matrix is a matrix of n x n size where each row and column represent a vertex on the graph. This matrix means that there is a point in the matrix that represents whether there is an edge between one vertex and another. This is indicated with a 1 for indicating that there is an edge, and a 0 to indicate that there is not.

Step1: Traverse the digraph, number the vertexes
Because it is an n x n matrix, a typical traversal will require n^2 number of steps to complete as the entire matrix must be scanned for the next starting point once a dead end is reached.
Step2: Reverse the digraph:
Reversing the digraph also involves going through the entire digraph as this requires transforming the matrix. This will also require n^2 steps.
Step3:Traverse the new digraph:
This is a traversal, and much like the first will also be n^2.

This means that the efficiency of algorithm is roughly 3n^2, better represented by the efficiency case theta(n^2)

Adjacency List:
An adjacency list represents a graph with a list for each vertex where the edges to other vertices are listed.

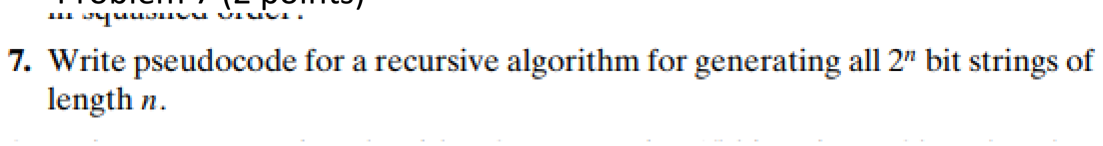
Step1:Traverse the digraph

Good news, the list is linear, which means that traversing it is also linear with efficiency n for number of vertices.
Step2: Reverse the digraph:
Reversing the digraph is also linear, however its length is based on the number of edges present instead of the number of edges. This means that the efficiency for this step is e for number of edges.
Step 3: Traverse the new digraph
It's the same as step , so also efficiency n

This means the total efficiency for an adjacency list is:
Theta(n+e) which is linear! Yay!

10) Exercises 4.3

a. Problem 7 (2 points)



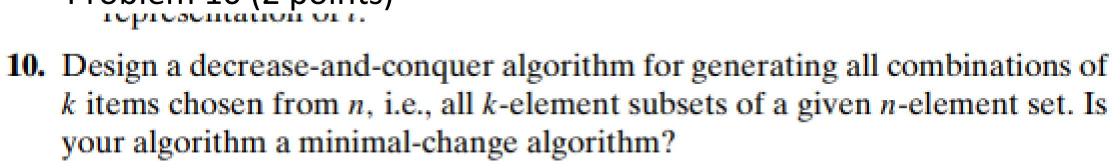
7. Write pseudocode for a recursive algorithm for generating all 2^n bit strings of length n.

This problem requires a recursive algorithm that generates all 2^n strings of length n, which essentially means it wants every single combination of 1s and 0s for a string with n decimal places. Sounds easy, we just keep adding 1 to a default array of 0s.

```
//the function BS (BitString) returns the bitstrings)
Def BS(n)
if n == 0)
return ['] // returns an empty string to add things to.
else
    PreviousString = BS(n-1)
    newstring =[]
    for b in PreviousString:
        b.append(0)
        b.append(1)
    return newstring
print(BS(n)) //the new array generated by the above code.
```

CSC 3110
Homework 2
May Wandyez
Gq5426
2/12/2024

b. Problem 10 (2 points)



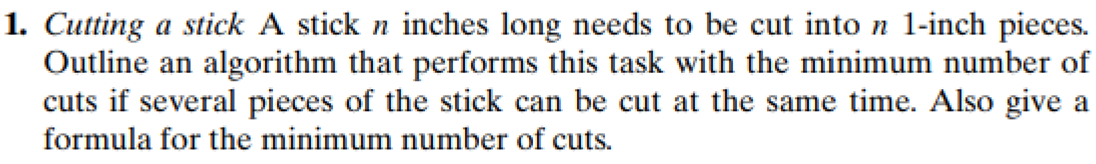
10. Design a decrease-and-conquer algorithm for generating all combinations of k items chosen from n, i.e., all k-element subsets of a given n-element set. Is your algorithm a minimal-change algorithm?

The given problem is that for a set of size n, generate all subsets of size k. Use a decrease and conquer algorithm to achieve the goal. I was lazy and designed a minimal change algorithm

```
Def GeneratorOfSubsetts(set,k):
        N = len(set)
//deal with the edge cases, these are important for recursion as these also
//represent the end of the recursion calls.
If k>n:
        Return [] //mission accomplished, k is bigger than all possible choices
Elif k==0
        Return [[]] //mission accomplished, the subset never existed
Elif k==n:
        Return[set]//mission accomplished, the set WAS the subset.

//generate all subsets with the first element included
//this is all subsets that are one size smaller than previous input
//these later get recursively combined
Sfe = GeneratorOfSubsets(set[1:];k-1)
//great, now put that first element back in to every subset we just made
Sfe = GeneratorOfSubsets( [set[0]] + subset for subset in sfe]

//generate all the subsets with the first element NOT included
Swfe = GeneratorOfSubets(set[1:],k)

//great, all subsets for the desired function can be expressed between
//the subsets without the first element, and the subsets without the first
//element so return that
return sfe + swfe
```

11) Exercises 4.4

a. Problem 1 (2 points)



1. *Cutting a stick* A stick n inches long needs to be cut into n 1-inch pieces. Outline an algorithm that performs this task with the minimum number of cuts if several pieces of the stick can be cut at the same time. Also give a formula for the minimum number of cuts.

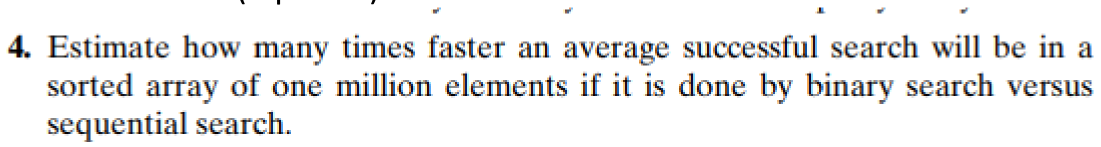
CSC 3110
Homework 2
May Wandyez
Gq5426
2/12/2024

We want to cut a stick that is n inches long into 1 inch pieces. We can also cut multiple pieces at the same time. This means we ought to try to maximize the number of pieces cut at the same time. Inherently, a cut separates an object into two pieces, which means in order to optimize the number of cuts we should optimize the size of each piece to be as large as possible, this can be done by cutting exactly in the middle.

Step 1: If the piece is uneven and not 1, cut off one inch or cut subtract floor(n). It is now even. Yay
Step 2: Sort all pieces that are 1 inch into the finished pile
Step 3: Align all pieces at the cutting point, so that each of their centers is aligned with each other they are ALL even so they can be indefinitely cut in the center until they are 1 inch
Step 4: Cut at the cutting point

The minimal number of cuts would be log2(n), as the formula is logarithmic due its bisections.

b. Problem 4 (2 points)


4. Estimate how many times faster an average successful search will be in a sorted array of one million elements if it is done by binary search versus sequential search.

Assuming the average search is randomly searching for an object in the array, the search time would be n/2, where n=1,000,000 -> 500,000
A binary search bisects, which means its search time would be log2(n) where n = 1000000 -> 19.9315

This would mean that a binary search would be 500000/19.9315 = 25085 times faster than a sequential search. Remember, though sequential search is slow, it ALWAYS works.