CSE 531: Algorithm Analysis and Design – Homework 1

Christopher Varghese | cgvarghe | 50411435

**Problem 1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | O | Ω | Θ |
| 2n | n2 | Yes | No | No |
| n2 | 4n2 | Yes | Yes | Yes |
| n3 | 4n2 + 5n | No | Yes | No |
| log2 n | n | Yes | No | No |
| n2 - 3n2 | 5n3 | Yes | No | No |
| n2 - 3n3 | n2 + n log2 n | Yes | No | No |
| (n + 1)! | n! + 100n3 | Yes | Yes | Yes |
|  | 2n | No | Yes | No |
| log3 n | log2(n20) | Yes | Yes | Yes |
| (n + log2 n)4 | (n2 + n log2 n)2 | Yes | Yes | Yes |

**Problem 2a**

Let and , prove .

Reducing :

* Apply L5

We now have the following:

Proof Approach (by definition):

Proving :

There exists a constant and , for any number , we have , this implies that , thus is in .

Proving :

There exists a constant and , for any number , we have , this implies that , thus is in .

Proving :

Therefore, is in since is in and is in

**Problem 2b**

Let and , prove .

Reducing :

* Apply S2 with , , ,
* Apply S1 with , ,
* Apply S3 with , , ,
* Apply S2 with , , ,
* Apply S3 with , , ,
* Apply S3 with , , ,
* Apply S2 with , , ,
* Apply S2 with , , ,
* Apply S1 with , ,
* Apply S8 with
* Apply S1 with , ,
* Apply S1 with , ,
* Simplify

We now have the following:

Proof Approach (by definition):

:

There exists a constant and , for any number , we have , this implies that , thus is in .

:

There exists a constant and , for any number , we have , this implies that , thus is in .

:

Therefore, is in since is in and is in

**Problem 2c**

Let and

Simplify:



:

*Proof by Definition*

There exists a constant and , for any number , we have , this implies that , thus is in .

:

*Proof by Contradiction*

Assume . By definition, there exists a constant and , for any number , for any number n , we have .

However, .

This derives a contradiction because for constant and , for any number such that holds. Thus, .

:

Therefore, since .

**A white rectangular object with black border

Description automatically generatedProblem 3a**

Proof of Correctness:

The algorithm’s outer loop iterates from to , while the inner loop iterates from to for each value of in , thus the algorithm will terminate after a finite number of iterations.

This guarantees the algorithm considers all pairs in and checks in every iteration of j, if the condition holds for at least one pair, it returns “yes” and terminates.

Upon completion of the loops, this indicates the algorithm did not find any pair in such that , hence it returns “no” and terminates.

Runtime Analysis:

The worst-case running time of the algorithm is as the outer loop iterates from to , and for each iteration of the outer loop, the inner loop iterates from to.

Therefore, if contains no pairs that satisfy the condition , the algorithm will complete all iterations of outer and inner loops before terminating.

Under these conditions, the algorithm must terminate in at most iterations.

**A screenshot of a computer

Description automatically generated**

Proof of Incorrectness:

Given an array of unsorted non-negative integers , the algorithm will start and will continually increase as the inputs will satisfy both conditions and until .

The algorithm will continue to iterate the outer loop from to as does not satisfy , hence the algorithm will return the result “no” and terminate.

However, the correct solution would recognize there exists a pair in that satisfies , therefore the correct solution would return “yes” and exit.

**A white box with black text

Description automatically generated**Proof of Incorrectness:

Given an array of unsorted non-negative integers , the algorithm will start and select , because the algorithm will eventually move to the next .

At , the algorithm selects , the conditions return , due to the algorithm’s failure to compare the last index with the first, it will eventually return the result “no” and exit.

However, the correct solution would recognize there exists a pair within that satisfies , therefore the correct solution would return “yes” and exit.

**A screenshot of a computer

Description automatically generatedProblem 3b**

Proof of Correctness:

The algorithm’s outer loop iterates from to , while the inner loop iterates from 1 to n until either or , thus the algorithm will terminate after a finite number of iterations.

This guarantees the algorithm considers all pairs in in increasing order using to adjust and check , thus if the condition holds for at least one pair, it returns “yes” and terminates.

Upon completion of the loops, this indicates the algorithm did not find any pair in such that , hence it returns “no” and terminates.

Runtime Analysis:

The worst-case running time of the algorithm is as the outer loop and inner loop iterates from to once each.

Thus, if contains no pairs that satisfy the condition , the algorithm will increment until the end of in attempt to find a pair that meets the condition.

Under these conditions, the algorithm must terminate in at most iterations.

**A white box with black text

Description automatically generated**Proof of Correctness:

The algorithm’s outer loop iterates from to , while the inner loop continues until is not met and then proceeds to the iteration, thus the algorithm will terminate after a finite number of iterations.

This guarantees the algorithm considers all pairs in that satisfy by adjusting based on its position at in , such that if the condition holds for at least one pair, it returns “yes” and terminates.

Upon completion of the loops, this indicates the algorithm did not find any pair in such that , hence returns “no” and terminates.

Runtime Analysis:

The worst-case running time of the algorithm is as the outer loop iterates from to , and the inner loop guarantees splitting and removing the search space by a factor of resulting in searches.

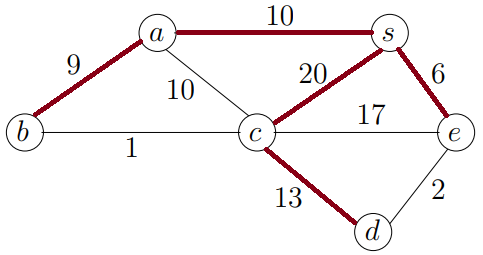
Therefore, if contains no pairs that satisfy the condition , the algorithm will perform at most searches for each th index.

Under these conditions, the algorithm must terminate in at most iterations.

**Problem 4a**

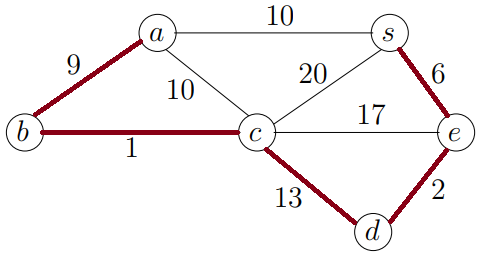
**A diagram of a graph

Description automatically generated**

BFS Execution:

**10(s-a) → 20(s-c) → 6(s-e) → 9(a-b) → 13(c-d)**

Sequence = {S, A, C, E, B, D}

DFS Execution:

**9(a-b) → 1(b-c) → 13(c-d) → 2(d-e) → 6(e-s)**

Sequence = {A, B, C, D, E, S}

**Problem 4b**

:

1:

2:

3:

4:

5:

6:

7:

8:

9:

10:

11:

12:

13:

Pseudocode for Graph Odd Cycle Detection:

1:

2:

3:

4:

5:

*Input: graph using an adjacency list representation*

*Output: whether there exists an odd cycle in or not*

Proof of Correctness:

The algorithm’s first loop iterates vertices once from to and the second loop iterates through each edge of once from to , thus the algorithm will terminate after a finite number of iterations.

This ensures the algorithm detects an odd cycle using bipartite coloring by asserting if a previously discovered neighbor exists with , thus detecting has an odd cycle, prints it to the user, and terminates.

Hence, if no odd cycle exists, the algorithm has assigned a bipartite coloring to all nodes and neighbors such that , the algorithm goes through all nodes of and confirms no odd cycle, prints it to the user, and terminates.

Runtime Analysis:

The worst-case runtime of the algorithm is as the algorithm goes through each vertex once in the adjacency list and iterates through all edges to visit each neighbor once.

If there is no odd cycle in , the algorithm will perform all vertex iterations and edge iterations leveraging a discovery map, color map, and queue of operations.

Under these conditions, the algorithm must terminate in at most iterations.