CSE 531: Algorithm Analysis and Design – Homework 1

Christopher Varghese | cgvarghe | 50411435

Problem 1

$f(n)$	$g(n)$	O	Ω	Θ
2n	n^2	Yes	No	No
n^2	$4n^2$	Yes	Yes	Yes
n^3	$4n^2 + 5n$	No	Yes	No
$\log_2 n$	n	Yes	No	No
$n^2 - 3n^2$	$5n^3$	Yes	No	No
$n^2 - 3n^3$	$n^2 + n \log_2 n$	Yes	No	No
(n + 1)!	$n! + 100n^3$	Yes	Yes	Yes
$n^{2\log_2 n}$	2^n	No	Yes	No
$\log_3 n$	$\log_2(n^{20})$	Yes	Yes	Yes
$(n + \log_2 n)^4$	$(n^2 + n \log_2 n)^2$	Yes	Yes	Yes

Problem 2a

Let $f(n) = 10n\, log_2(2^n) + 0.2n$ and $g(n) = n^2$, prove $f(n) \in \Theta(g(n))$.

Reducing $f(n)$:

- Apply L5

$$10n\, log(2^n) + 0.2n = 10n\, 2^{log(n)} + 0.2n = 10n * n + 0.2n = 10n^2 + 0.2n$$

We now have the following:

$$f(n) = 10n^2 + 0.2n$$

$$g(n) = n^2$$

<u>Proof Approach (by definition):</u>

Proving $f(n) = O(g(n))$:

There exists a constant $c_1 = 15$ and $n_0 = 50$, for any number $n \geq n_0$, we have $10n^2 + 0.2n < 12n^2$, this implies that $f(n) = 10n^2 + 0.2n \leq c_1 * g(n)$, thus $f(n)$ is in $O(g(n))$.

■

Proving $f(n) = \Omega(g(n))$:

There exists a constant $c_1 = 1$ and $n_0 = 50$, for any number $n \geq n_0$, we have $10n^2 + 0.2n > n^2$, this implies that $f(n) = 10n^2 + 0.2n \geq c_1 * g(n)$, thus $f(n)$ is in $\Omega(g(n))$.

■

Proving $f(n) = \Theta(g(n))$:

Therefore, $f(n)$ is in $\Theta(g(n))$ since $f(n)$ is in $O(g(n))$ and $f(n)$ is in $\Omega(g(n))$.

■

Problem 2b

Let $f(n) = \sum_{i=1}^{n}(1 + \sum_{j=i}^{n} n)$ and $g(n) = n^3$, prove $f(n) \in \Theta(g(n))$.

Reducing $f(n)$:

- Apply S2 with $c = n, f(i) = 1, j = i, k = n$

$$\sum_{i=1}^{n}\left(1+\sum_{j=i}^{n} n\right) = \sum_{i=1}^{n}\left(1+n\sum_{j=i}^{n} 1\right)$$

- Apply S1 with $j = i, k = n, c = 1$

$$\sum_{i=1}^{n}\left(1+n\sum_{j=i}^{n} 1\right) = \sum_{i=1}^{n}(1+n(n-i+1))$$

- Apply S3 with $f(i) = 1, g(i) = n(n-i+1), j = 1, k = n$

$$= \sum_{i=1}^{n} 1 + \sum_{i=1}^{n} n(n-i+1)$$

- Apply S2 with $c = n, f(i) = n - i + 1, j = 1, k = n$

$$= \sum_{i=1}^{n} 1 + n\sum_{i=1}^{n} n-i+1$$

- Apply S3 with $f(i) = n - i, g(i) = 1, j = 1, k = n$

$$= \sum_{i=1}^{n} 1 + n\left(\sum_{i=1}^{n} n-i+\sum_{i=1}^{n} 1\right)$$

- Apply S3 with $f(i) = n, g(i) = -i, j = 1, k = n$

$$= \sum_{i=1}^{n} 1 + n\left(\sum_{i=1}^{n} n+\sum_{i=1}^{n} -i+\sum_{i=1}^{n} 1\right)$$

- Apply S2 with $c = -1, f(i) = i, j = 1, k = n$

$$= \sum_{i=1}^{n} 1 + n\left(\sum_{i=1}^{n} n-\sum_{i=1}^{n} i+\sum_{i=1}^{n} 1\right)$$

- Apply S2 with $c = n, f(i) = 1, j = 1, k = n$

$$= \sum_{i=1}^{n} 1 + n\left(n\sum_{i=1}^{n} 1-\sum_{i=1}^{n} i+\sum_{i=1}^{n} 1\right)$$

- Apply S1 with $j = 1, k = n, c = 1$

$$= \sum_{i=1}^{n} 1 + n\left(n(n-1+1)-\sum_{i=1}^{n} i+\sum_{i=1}^{n} 1\right)$$

- Apply S8 with $k = n$

$$= \sum_{i=1}^{n} 1 + n\left(n(n-1+1) - \frac{n(n+1)}{2} + \sum_{i=1}^{n} 1\right)$$

- Apply S1 with $j = 1, k = n, c = 1$

$$= \sum_{i=1}^{n} 1 + n\left(n(n-1+1) - \frac{n(n+1)}{2} + (n-1+1)\right)$$

- Apply S1 with $j = 1, k = n, c = 1$

$$= (n-1+1) + n\left(n(n-1+1) - \frac{n(n+1)}{2} + (n-1+1)\right)$$

- Simplify

$$= n + n\left(n(n) - \frac{n(n+1)}{2} + n\right)$$

$$= n + n\left(n^2 - \frac{n^2+n}{2} + n\right)$$

$$= n + n^3 - \frac{n^3+n^2}{2} + n^2$$

$$= n^3 - \frac{n^3+n^2}{2} + n^2 + n$$

$$= \frac{2n^3}{2} - \frac{n^3+n^2}{2} + \frac{2n^2}{2} + \frac{2n}{2}$$

$$= \frac{2n^3}{2} - \frac{n^3}{2} - \frac{n^2}{2} + \frac{2n^2}{2} + \frac{2n}{2}$$

$$= \frac{n^3}{2} + \frac{n^2}{2} + \frac{2n}{2}$$

$$= \frac{n^3+n^2+2n}{2}$$

We now have the following:

$$f(n) = \frac{n^3 + n^2 + 2n}{2}$$

$$g(n) = n^3$$

<u>Proof Approach (by definition):</u>

$f(n) = O(g(n))$:

There exists a constant $c_1 = 1$ and $n_0 = 50$, for any number $n \geq n_0$, we have $\frac{n^3+n^2+2n}{2} \leq n^3$, this implies that $f(n) = \frac{n^3+n^2+2n}{2} \leq c_1 * g(n)$, thus $f(n)$ is in $O\big(g(n)\big)$.

■

$f(n) = \Omega(g(n))$:

There exists a constant $c_1 = \frac{1}{3}$ and $n_0 = 50$, for any number $n \geq n_0$, we have $\frac{n^3+n^2+2n}{2} \geq \frac{n^3}{3}$, this implies that $f(n) = \frac{n^3+n^2+2n}{2} \geq c_1 * g(n)$, thus $f(n)$ is in $\Omega\big(g(n)\big)$.

■

$f(n) = \Theta(g(n))$:

Therefore, $f(n)$ is in $\Theta\big(g(n)\big)$ since $f(n)$ is in $\mathrm{O}\big(g(n)\big)$ and $f(n)$ is in $\Omega\big(g(n)\big)$.

■

Problem 2c

Let $f(n) = 2^{2n+10\sqrt{n}+4}$ and $g(n) = 5^n$

Simplify:

- $2^{2n+10\sqrt{n}+4} = \log\left(2^{2n+10\sqrt{n}+4}\right) = 2n + 10\sqrt{n} + 4$
- $5^n = \log(5^n) = n * \log_2(5) \approx 2.32n$

$f(n) = O(g(n))$:

Proof by Definition

There exists a constant $c_1 = 2^{100}$ and $n_0 = 50$, for any number $n \geq n_0$, we have $2^{2n+10\sqrt{n}+4} \leq 2^{100} * 5^n$, this implies that $f(n) = 2^{2n+10\sqrt{n}+4} \leq c_1 * g(n)$, thus $f(n)$ is in $O\big(g(n)\big)$.

■

$f(n) \neq \Omega\big(g(n)\big)$:

Proof by Contradiction

Assume $2^{2n+10\sqrt{n}+4} = \Omega(5^n)$. By definition, there exists a constant $c_1 > 0$ and $n_0 > 0$, for any number $n \geq n_0$, for any number n $n \geq n_0$, we have $f(n) = 2^{2n+10\sqrt{n}+4} \geq c_1 * g(n)$.

However, $2^{2n+10\sqrt{n}+4} \geq c_1 * 5^n = \log\left(2^{2n+10\sqrt{n}+4}\right) \geq \log(c_1) + \log(5^n) = 2n + 10\sqrt{n} + 4 \geq \log_2(c_1) + n * \log_2(5) = \frac{2n+10\sqrt{n}+4}{n*\log_2(5)} \geq \log_2(c_1)$.

This derives a contradiction because for constant $c_1 = 10$ and $n' = 100$, for any number $n \geq n_0$ such that $\frac{2n\prime+10\sqrt{n\prime}+4}{n\prime*\log_2(5)} < \log_2(c_1)$ holds. Thus, $2^{2n+10\sqrt{n}+4} \neq \Omega(5^n)$.

■

$f(n) \neq \Theta\big(g(n)\big)$:

Therefore, $f(n) \neq \Theta\big(g(n)\big)$ since $2^{2n+10\sqrt{n}+4} \neq \Omega(5^n)$.

■

Problem 3a

Algorithm 1

for $i \leftarrow 1$ **to** n **do**
    for $j \leftarrow 1$ **to** n **do**
        if $A[j] = (A[i])^2$ **then**
            return "yes"
return "no"

Proof of Correctness:

The algorithm's outer loop iterates i from 1 to n, while the inner loop iterates j from 1 to n for each value of i in A, thus the algorithm will terminate after a finite number of iterations.

This guarantees the algorithm considers all (i, j) pairs in A and checks $A[j] = (A[i])^2$ 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 (i, j) pair in A such that $A[j] = (A[i])^2$, hence it returns "no" and terminates.

■

Runtime Analysis:

The worst-case running time of the algorithm is $\boldsymbol{O(n^2)}$ as the outer loop iterates from 1 to n, and for each iteration of the outer loop, the inner loop iterates from 1 to n.

Therefore, if A contains no pairs that satisfy the condition $A[j] = (A[i])^2$, the algorithm will complete all iterations of outer and inner loops before terminating.

Under these conditions, the algorithm must terminate in at most n^2 iterations.

■

Algorithm 2

$j \leftarrow 1$
for $i \leftarrow 1$ **to** n **do**
    while $j \leq n$ **and** $A[j] < (A[i])^2$ **do**
        $j \leftarrow j + 1$
    if $j \leq n$ **and** $A[j] = (A[i])^2$ **then**
        return "yes"
return "no"

Proof of Incorrectness:

Given an array of n unsorted non-negative integers $A = [200, 1, 99, 4]$, the algorithm will start $i = 1$ and will continually increase j as the inputs will satisfy both conditions $j \leq n$ and $A[j] < (A[i])^2$ until $j = 5$.

The algorithm will continue to iterate the outer loop from $i + 1$ to n as j does not satisfy $j \leq n = 5 \leq 4$, hence the algorithm will return the result "no" and terminate.

However, the correct solution would recognize there exists a pair $(i = 2, j = 2)$ in A that satisfies $A[j] = (A[i])^2 \rightarrow 1 = (1)^2 \rightarrow 1 = 1$, therefore the correct solution would return "yes" and exit.

■

Algorithm 3

```
for i ← 1 to n do
    ℓ ← i, r ← n
    while ℓ ≤ r do
        j ← ⌈(ℓ+r)/2⌉
        if A[j] = (A[i])^2 then
            return "yes"
        else if (A[i])^2 < A[j] then
            r ← j − 1
        else
            ℓ ← j + 1
return "no"
```

Proof of Incorrectness:

Given an array of n unsorted non-negative integers $A = [4, 2]$, the algorithm will start $i = 1$ and select $j = \left\lceil \frac{l+r}{2} \right\rceil = \left\lceil \frac{1+2}{2} \right\rceil = 2$, because $A[j] = (A[i])^2 \rightarrow 2 \neq 16$ the algorithm will eventually move to the next i.

At $i = 2$, the algorithm selects $j = \left\lceil \frac{l+r}{2} \right\rceil = \left\lceil \frac{2+2}{2} \right\rceil = 2$, the conditions return $A[j] = (A[i])^2 \rightarrow 2 \neq 4$, 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 $(i = 2, j = 1)$ within A that satisfies $A[j] = (A[i])^2 \rightarrow 4 = (2)^2 \rightarrow 4 = 4$, therefore the correct solution would return "yes" and exit.

■

Problem 3b

Algorithm 2

$j \leftarrow 1$
for $i \leftarrow 1$ **to** n **do**
    while $j \leq n$ **and** $A[j] < (A[i])^2$ **do**
        $j \leftarrow j + 1$
    if $j \leq n$ **and** $A[j] = (A[i])^2$ **then**
        return "yes"
return "no"

Proof of Correctness:

The algorithm's outer loop iterates i from 1 to n, while the inner loop iterates j from 1 to n until either $j > n$ or $A[j] > (A[i])^2$, thus the algorithm will terminate after a finite number of iterations.

This guarantees the algorithm considers all (i, j) pairs in A in increasing order using j to adjust and check $A[j] = (A[i])^2$, 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 (i, j) pair in A such that $A[j] = (A[i])^2$, hence it returns "no" and terminates.

$\blacksquare$

Runtime Analysis:

The worst-case running time of the algorithm is $\boldsymbol{O(n+n) = O(2n) = O(n)}$ as the outer loop and inner loop iterates from 1 to n once each.

Thus, if A contains no pairs that satisfy the condition $A[j] = (A[i])^2$, the algorithm will increment j until the end of A in attempt to find a pair that meets the condition.

Under these conditions, the algorithm must terminate in at most $2n$ iterations.

$\blacksquare$

Algorithm 3

```
for i ← 1 to n do
    ℓ ← i, r ← n
    while ℓ ≤ r do
        j ← ⌈(ℓ+r)/2⌉
        if A[j] = (A[i])^2 then
            return "yes"
        else if (A[i])^2 < A[j] then
            r ← j − 1
        else
            ℓ ← j + 1
return "no"
```

Proof of Correctness:

The algorithm's outer loop iterates i from 1 to n, while the inner loop continues until $l \leq r$ is not met and then proceeds to the $i+1$ iteration, thus the algorithm will terminate after a finite number of iterations.

This guarantees the algorithm considers all (i, j) pairs in A that satisfy $A[j] = (A[i])^2$ by adjusting j based on its position at $\left\lceil \frac{l+r}{2} \right\rceil$ in A, 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 (i, j) pair in A such that $A[j] = (A[i])^2$, hence returns "no" and terminates.

$\blacksquare$

Runtime Analysis:

The worst-case running time of the algorithm is $\boldsymbol{O(nlogn)}$ as the outer loop iterates from 1 to n, and the inner loop guarantees splitting and removing the search space by a factor of 2 resulting in $\log_2(n)$ searches.

Therefore, if A contains no pairs that satisfy the condition $A[j] = (A[i])^2$, the algorithm will perform at most $O(logn)$ searches for each ith index.

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

$\blacksquare$

Problem 4a

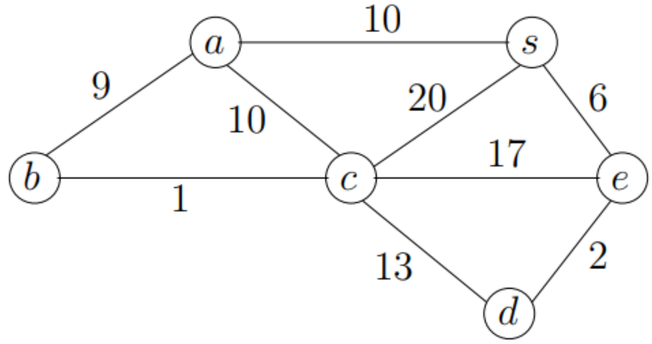
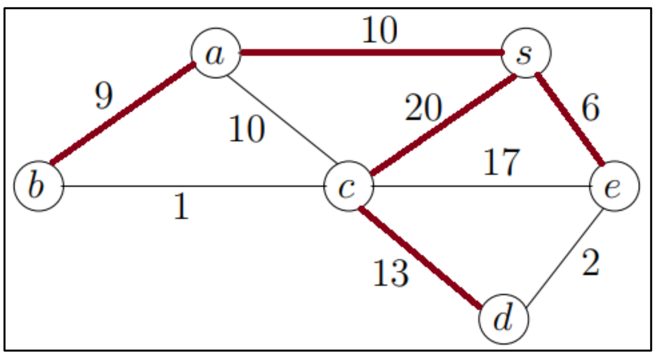


Figure 1: Traverse the graph using BFS and DFS

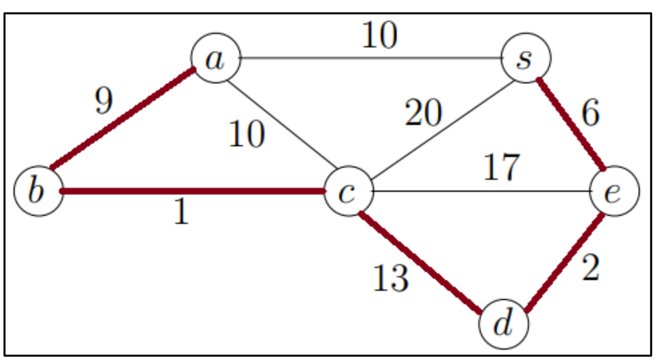
BFS Execution:



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

Pseudocode for Graph Odd Cycle Detection:

```
test_odd_cycle(s, adj_list, discovered):
    initiliaze queue and color
    head ← 1, tail ← 1, queue[1] ← s
    discovered[s] ← True
    color[s] ← 0
    while head ≤ tail do:
        v ← queue[head], head ← head + 1
        for all u of adj_list[v] do:
            if u is not in discovered then:
                tail ← tail + 1, queue[tail] = u
                discovered[u] ← True
                color[u] ← 1 − color[v]
            else if color[u] = color[v]:
                print("odd cycle detected") and exit
```

```
does_my_graph_have_an_odd_cycle(adj_list):
    initialize discovered
    for each vertex v ∈ adj_list do:
        if v is not in discovered then:
            test_odd_cycle(v, adj_list, discovered)
    print("no odd cycle detected") and exit
```

Input: graph $G = (V, E)$ using an adjacency list representation

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

Proof of Correctness:

The algorithm's first loop iterates vertices v once from 1 to n and the second loop iterates through each edge m of v once from 1 to m, 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 u exists with $color[u] = color[v]$, thus detecting G 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 v and neighbors u such that $color[u] \neq color[v]$, the algorithm goes through all nodes of V and confirms no odd cycle, prints it to the user, and terminates.

■

Runtime Analysis:

The worst-case runtime of the algorithm is $\boldsymbol{O(n+m)}$ as the algorithm goes through each vertex v once in the adjacency list and iterates through all m edges to visit each neighbor u once.

If there is no odd cycle in G, the algorithm will perform all n vertex iterations and m edge iterations leveraging a discovery map, color map, and queue of $O(1)$ operations.

Under these conditions, the algorithm must terminate in at most $n + m$ iterations.

■