# Homework 1

September 13, 2025

## 1 [BvG 1.5]

Show that $\lceil \lg(n+1) \rceil = \lfloor \lg n \rfloor + 1$ for integers $n \geq 1$.

*Hint*: Group values of $n$ into ranges of the form $2^k \leq n < 2^{k+1}$.

**Answer:**

We are given: $2^k \leq n < 2^{k+1}$

$lg2^k \leq lgn < lg2^{k+1}$

$k \leq lgn < k+1$

We can get floor of $lgn$ from here:

$\lfloor \lg n \rfloor = k$

Now we need to show: $\lceil \lg(n+1) \rceil = k+1$

Shifting Inequalities for: $n < 2^{k+1}$

we have: $n+1 \leq 2^{k+1}$

Log on both sides: $lg(n+1) \leq k+1$

Combined: $k < lg(n+1) \leq k+1$

then, $\lceil \lg(n+1) \rceil = k+1$

Finally, $\lceil \lg(n+1) \rceil = k+1 = \lfloor \lg n \rfloor + 1$

**Outcomes:** CS 6, 5870-1

## 2 [GT R-1.2]

Show that the MaxSubSlow algorithm runs in $\Omega(n^3)$ time.

**Answer:**

1. Innermost loop (runs i from j to k): The $s \leftarrow s + A[i]$ runs k-j+1 times.

So, $f(j,k) = k - j + 1$

2. Middle loop (runs k from j to n): We sum up f(j,k):

$$g(j) = \sum_{k=j}^{n} f(j,k) = \sum_{k=j}^{n} (k - j - 1)$$

Let $x = k - j$, so when k goes from j to n, x goes from 0 to n-j:

$$g(j) = \sum_{x=0}^{n-j} (x+1) = \sum_{x=1}^{n-j+1} x = \frac{(n-j+1)(n-j+2)}{2}$$

3. Outer loop (runs j from 1 to n):

$$T(n) = \sum_{j=1}^{n} g(j) = \sum_{j=1}^{n} \frac{(n-j+1)(n-j+2)}{2}$$

Now let a = n-j+1, j goes from 1 to n, a goes from n down to 1:

$$T(n) = \sum_{a=1}^{n} = \frac{a(a+1)}{2} = \frac{1}{2} \sum_{a=1}^{n} (a^2 + a)$$

$$= \frac{1}{2} \left( \frac{n(n+1)(2n+1)}{6} + \frac{n(n+1)}{2} \right)$$

$$= \frac{n(n+1)}{2} \cdot \frac{1}{2} \left( \frac{2n+1}{3} + 1 \right) = \frac{n(n+1)(2n+4)}{12}$$

$$\frac{n(n+1)(n+2)}{6}$$

$$T(n) = \frac{n^3 + 3n^2 + 2n}{6}$$

Here we have the leading term as $\frac{n^3}{6}$, we can say: $T(n) \in \omega(n^3)$

**Outcomes:** CS 2, CS 6, 5870-1

# 3 [GT R-1.7]

Order the following list of functions by the $O(\,)$ notation, from smallest to largest. Group together those functions that are in $\Theta(\,)$ of one another.

| | | | | |
|---|---|---|---|---|
| $6n \lg n$ | $2^{100}$ | $\log \log n$ | $\log^2 n$ | $2^{\log n}$ |
| $2^{2^n}$ | $\lceil \sqrt{n} \rceil$ | $n^{0.01}$ | $1/n$ | $4n^{3/2}$ |
| $3n^{0.5}$ | $5n$ | $\lfloor 2n \log^2 n \rfloor$ | $2^n$ | $n \log_4 n$ |
| $4^n$ | $n^3$ | $n^2 \log n$ | $4^{\log n}$ | $\sqrt{\log n}$ |

*Note*: If no base is given, assume that $\log n$ is $\log_2 n$.

*Hint*: When in doubt about two functions $f(n)$ and $g(n)$, consider $\log f(n)$ and $\log g(n)$ or $2^{f(n)}$ and $2^{g(n)}$.

**Answer:**

1. $1/n$
2. $2^{100} : \Theta(1)$
3. $\log \log n : \Theta(\log \log n)$
4. $\sqrt{\log n} : \Theta(\log n)$
5. $\log^2 n : \Theta(\log n)$
6. $n^{0.01} : \Theta(n^{0.01})$
7. $\lceil \sqrt{n} \rceil$, $3n^{0.5} : \Theta(n^{1/2})$
8. $5n : \Theta(n)$
9. $2^{\log n} : \Theta(n)$
10. $6n \lg n$, $n \log_4 n : \Theta(n \log n)$
11. $\lfloor 2n \log^2 n \rfloor : \Theta(n \log^2 n)$
12. $4n^{3/2} : \Theta(n^{3/2})$
13. $4^{\log n} : \Theta(n^2)$
14. $n^2 \log n : \Theta(n^2 \log n)$
15. $n^3 : \Theta(n^3)$
16. $2^n : \Theta(2^n)$
17. $2^{2^n}$, $4^n : \Theta(4^n)$

**Outcomes:** CS 6, 5870-1

# 4 [GT C-1.14]

A $n$-degree polynomial $p(x)$ is an equation of the form

$$p(x) = \sum_{i=0}^{n} a_i x^i$$

where $x$ is a real number and each $a_i$ is a constant.

**a** Describe a simple $O(n^2)$-time method for computing $p(x)$ for a particular value of $x$.

**b** Consider now a rewriting of $p(x)$ as

$$p(x) = a_0 + x\,(a_1 + x\,(a_2 + x\,(a_3 + \cdots + x\,(a_{n-1} + x a_n)\cdots)))$$

which is known as *Horner's method*. Using the $O(\,)$ notation, characterize the number of multiplications and additions this method of evaluation uses.

* **Answer:**

**a)** $O(n^2)$ Method: A brute force method is to compute each term $a_i x^i$ individually and sum them up. The term $a_i x^i$ can be computed by multiplying $x$ by itself $i$ times. Here we have nested loop structure.

p(x) = $a_0 x^0 + a_1 x^1 + \ldots + a_n x^n$

**Algorithm 1** Simple $O(n^2)$ Polynomial Evaluation

```
result ← 0
for i ← 0 to n do
    termPower ← 1
    for j ← 1 to i do
        termPower ← termPower · x
    end for
    termTotal ← a_i · termPower
    result ← result + termTotal
end for
return result
```

**b)** Horner's Method ($O(n)$): Horner's method is a more efficient way to evaluate the polynomial. It evaluates the nested form of the equation from the inside out. **(b) Horner's method ($O(n)$):**

**Algorithm 2** Horner's Method

```
result ← a_n
for i ← n − 1 downto 0 do
    result ← a_i + x · result
end for
return result
```

We start with $result \leftarrow a_n$ We iterate backwards, $i = n - 1$ to 0:

In every single iteration, we do:
1 multiplication: $x \cdot result$
1 addition: $a_i + ...$

That means we do $n$ multiplication and $n$ addition for $n$ degree polynomial.

Total work = n + n = 2n which we can say : O(n)

**Outcomes:** CS 2, CS 6, 5870-1, 5870-2

# 5 [GT C-1.19]

An array $A$ contains $n-1$ unique integers in the range $[0, n-1]$; that is, there is one number from this range that is not in $A$. Design an $O(n)$-time algorithm for finding that number. You are allowed to use only $O(1)$ additional space besides the array $A$ itself.

**Answer:**

**Algorithm 3** Algorithm notInArratRange ($A$):

**Require:** Input: An array with $n-1$ unique integers in the range $[0, n-1]$, and a number not in that range

**Ensure:** Output: Number not in range $[0, n-1]$.

```
rangeSum ← 0
arraySum ← 0
res ← 0
rangeSum ← n(n − 1)/2
for i ← 0 to n − 1 do
    arraySum ← arraySum + A[i]
end for
res ← arraySum − rangeSum
return res
```

**Outcomes:** CS 1, CS 2, 5870-1

# 6 [GT C-4.2]

The *Fibonacci sequence* is the sequence of numbers

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots$$

defined by the base cases $F_0 = 0$ and $F_1 = 1$ and the general-case recursive definition $F_k = F_{k-1} + F_{k-2}$ for $k \geq 2$. Show by induction that, for $k \geq 3$,

$$F_k \geq \phi^{k-2}$$

where $\phi = (1 + \sqrt{5})/2 \approx 1.618$, which is the well-known *golden ratio* that traces its history to the ancient Greeks.

*Hint*: Note that $\phi^2 = \phi + 1$; hence, $\phi^k = \phi^{k-1} + \phi^{k-2}$ for $k \geq 3$.

**Answer:**

Given: $F_k = F_{k-1} + F_{k-2}$ for $k \geq 2$

To Prove: $F_k \geq \phi^{k-2}$ for $k \geq 3$

Base Case: k = 3

$F_3 \geq \phi^{3-2}$

$2 \geq 1.618$ — $F_2 = 2$ from [0,1,1,2,3,...] and $\phi \approx 1.618$

Now, assume for all integers n with $3 \leq n \leq k$

We have: $F_n \geq \phi^{n-2}$

We have show that: $F_{n+1} \geq \phi^{n-1}$

$F_{n+1} = F_n + F_{n-1}$ — From Fibonacci

$F_{n+1} = F_n + F_{n-1} \geq \phi^{k-2} + \phi^{k-3}$

$F_{n+1} \geq \phi^{k-3}(\phi^1 + 1)$

$F_{n+1} \geq \phi^{k-3}(\phi^2)$ — From $\phi^2 = \phi + 1$

$F_{n+1} \geq \phi^{k-1}$

Hence, proved

**Outcomes:** CS 6, 5870-1

# 7 [SW 1.4.6]

Give the $\Theta(\cdot)$ order of growth (as a function of $N$) of the running times of each of the following code fragments:

**a.**
```
int sum=0;
for (int n=N; n > 0; n /= 2)
for (int i=0; i < n; i++)
sum++;
```

**b.**
```
int sum=0;
for (int i=1; i < N; i *= 2)
for (int j=0; j < i; j++)
sum++;
```

**c.**
```
int sum=0;
for (int i=1; i < N; i *= 2)
for (int j=0; j < N; j++)
sum++;
```

**Answer:**

a Outer loop: Runs $\lg N + 1$ times since we half $n$ each iteration. n = N, N/2, N/4,..1

- Inner Loop: Runs from 0 to n times. The value of n is updated in each outer loop iterations.
- $T(N) = N + \frac{N}{2} + \frac{N}{4} + \cdots + 1$
- $N(1 + \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{N})$
- This is geometric series with ratio $r = \frac{1}{2}$
- $N.(\frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{N}) \approx 2N$
- Overall: $\Theta(2N) = \Theta(N)$

b Outer loop: $\lg N$ since we double the index $i$ each iteration. $i = 1 + 2 + 4 + 8...2^k < N$

- Inner Loop: Runs from 0 to i times. The value of i is updated in each outer loop iterations.
- $T(N) = 1 + 2 + 4 + 8 + \cdots + 2^k$
- This is geometric series with a ration $r = 2$
- $2.(\frac{N}{2}) - 1 = N - 1$
- Overall: $\Theta(N)$

c Outer loop: $\lg N$ since we double the index $i$ each iteration. $i = 1 + 2 + 4 + 8...2^k < N$

- Inner Loop: Runs from 0 to N = N times independent of outer loop.
- Overall: $lgN.N = \Theta(NlgN)$

**Outcomes:** CS 1, CS 6, 5870-1

# 8 [SW 1.4.16]

*One-dimensional closest pair*. Write an algorithm that, given an array of $N$ numbers, finds a *closest pair*: two values whose difference is no greater than the difference of any other pair (in absolute value). The running time of your algorithm must be log-linear in the worst case.

**Answer:**

**Algorithm 4** One-dimensional closest pair

**Require:** An array A with N numbers
**Ensure:** Closest Pair (value1, value2)
  Array A is sorted in ascending order
  $value1 \leftarrow A[0]$
  $value2 \leftarrow A[1]$
  $diff \leftarrow |value2 - value1|$
  **for** $i \leftarrow 1$ to $n - 2$ **do**
    **if** $|A[i+1] - A[i]| < diff$ **then**
      $value1 \leftarrow A[i]$
      $value1 \leftarrow A[i+1]$
    **end if**
  **end for**
  **return** $value1$, $value2$

**Outcomes:** CS 2, 5870-1

# 9 [CLRS 3.2-3]

Prove that $\lg(n!) \in \Theta(n \lg n)$. Also prove that $n! \in \omega(2^n)$ and $n! \in o(n^n)$. *Hint*: Use Stirling's approximation: $n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1 + \Theta\left(\frac{1}{n}\right)\right)$

**Answer: a**

Given: $n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1 + \Theta\left(\frac{1}{n}\right)\right)$

Taking the log: $lg(n!) = lg(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1 + \Theta\left(\frac{1}{n}\right)\right))$

$lg(n!) = \frac{1}{2}lg(2\pi n) + nlg(\frac{n}{e}) + lg(1 + \Theta\left(\frac{1}{n}\right))$

$lg(n!) = \frac{1}{2}lg(2\pi n) + nlg(n) - nlg(e) + lg(1 + \Theta\left(\frac{1}{n}\right))$

Analyze as $n_{n\to\infty}$

Taking the limit: $lim_{n\to\infty}\frac{lg(n)!}{nlgn} = lim_{n\to\infty}\frac{\frac{1}{2}lg(2\pi n)+nlg(n)-nlg(e)+lg(1+\Theta\left(\frac{1}{n}\right))}{nlgn}$

$lim_{n\to\infty}\frac{\frac{1}{2}lg(2\pi n)}{nlgn} + \frac{nlgn}{nlgn} - \frac{nlg(e)}{nlgn} + \frac{lg(1+\Theta\left(\frac{1}{n}\right))}{nlgn}$

$lim_{n\to\infty}0 + 1 - 0 + 0$ — $[\frac{lg(1+\Theta\left(\frac{1}{n}\right))}{nlgn}]$: $\frac{lg(1)}{nlgn} : 0$

As $n \to \infty$ : $\frac{lg(2\pi n)}{nlgn}$, and $\frac{nlg(e)}{nlgn}$ approaches to 0

Therefore, limit : $0 + 1 - 0 = 1$

With positive limit we can say that $\lg(n!) \in \Theta(n \lg n)$

**Answer: b**

Prove $n! \in \omega(2^n)$

$lim_{n\to\infty}\frac{2^n}{n!} = lim_{n\to\infty}\frac{2^n}{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\Theta\left(\frac{1}{n}\right)\right)}$

$lim_{n\to\infty}\frac{1}{\sqrt{2\pi n}}\cdot\frac{2^n}{\left(\frac{n}{e}\right)^n}\cdot\frac{1}{\left(1+\Theta\left(\frac{1}{n}\right)\right)}$

$lim_{n\to\infty}\frac{1}{\sqrt{2\pi n}}\cdot\left(\frac{2e}{n}\right)^n\cdot\frac{1}{\left(1+\Theta\left(\frac{1}{n}\right)\right)}$

As $n \to \infty$ these terms: $\frac{2e}{n}$, $\frac{1}{\sqrt{2\pi n}}$, $\frac{1}{\left(1+\Theta\left(\frac{1}{n}\right)\right)}$ approach to 0

Therefore: $lim_{n\to\infty}\frac{2^n}{n!} = 0$ . Hence, $n! \in \omega(2^n)$

**Answer: c**

Prove $n! \in o(n^n)$

$lim_{n\to\infty}\frac{n!}{n^n} = lim_{n\to\infty}\frac{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\Theta\left(\frac{1}{n}\right)\right)}{n^n}$

$lim_{n\to\infty}\frac{\sqrt{2\pi n}.n^n\left(1+\Theta\left(\frac{1}{n}\right)\right)}{n^n.e^n}$

$lim_{n\to\infty}\frac{\sqrt{2\pi n}\left(1+\Theta\left(\frac{1}{n}\right)\right)}{e^n}$

$lim_{n\to\infty}\sqrt{2\pi n}\left(1+\Theta\left(\frac{1}{n}\right)\right).\frac{1}{e^n}$

Therefore, as $n\to\infty$ : the $e^n$ grows much faster than $\sqrt{n}$.

Denominator grows faster than the numerator. Hence, limit is 0

So that: $n!\in o\left(n^n\right)$

**Outcomes:** CS 6, 5870-1

# 10 [BvG 3.6 (modified)]

In this exercise, all integers are considered to be nonnegative, for simplicity. A *divisor* of an integer $k$ is any integer $d \neq 0$ such that $k/d$ has no remainder. A *common divisor* for a set of integers is an integer that is a divisor for each integer in the set. Euclid's algorithm for finding the greatest common divisor (GCD) of two nonnegative integers, $m$ and $n$, can be written as follows:

```
1:  procedure GCD(int m,int n)
2:      if n = 0 then
3:          answer ← m
4:      else if m < n then
5:          answer ← GCD(n, m)
6:      else
7:          r ← m − n · ⌊m/n⌋                    ▷ r is the remainder of m/n
8:          answer ← GCD(n, r)
9:      end if
10:     return answer
11: end procedure
```

The preconditions for $\text{GCD}(m, n)$ are that $m \geq 0, n \geq 0$ and $m + n > 0$. Prove the following using induction.

**a.** If the preconditions of $\text{GCD}(m, n)$ are satisfied, then the value that the function returns is *some* common divisor of $m$ and $n$.

**b.** If the preconditions of $\text{GCD}(m, n)$ are satisfied, then the value that the function returns is the *greatest* common divisor of $m$ and $n$.

*Hints*: If $d$ is a divisor of $k$, how can you rewrite $k$ in terms of $d$? How do you show that two sets are equal?

**Answer:**

Given preconditions: $m \geq 0, n \geq 0$ and $m + n > 0$

**a.** Base case: $m > 0$ and $n = 0$ . $GCD(m, 0) \leftarrow m : line2$. We can say that m is the common divisor since we can divide m and 0 by m. If (m,n) = GCD(1,0), then it returns m = 1 which divides both 0 and 1

**b.** Base case: $m = 0$ and $n > 0$ . $GCD(0, n) \leftarrow GCD(n, 0)$ since $m < n$ : line 4. We can say that n is the common divisor since we can divide 0 and n by n. If (m,n) = GCD(0,1), since $m < n$ so we call GCD(1,0) then it returns 1 which divides both 0 and 1

**c.** Therefore, the returned value is common divisor in above cases.

Let s = m + n for a pair (m,n). Now assume the claim holds for every pair (x,y) with $x + y < s$.

**a.** Case: $m < n$ calls GCD(n,m) which reduces the

**b.** Case $m \geq n > 0$. We call GCD(n,r), where $r = m - n.\lfloor \frac{m}{n} \rfloor$. The remainder r is: $0 \leq r < n$. Then $n + r \leq m + n = s$. Let's say value returned from GCD(n,r) = d which divides both n and r. This means $d \mid n$ and $d \mid r$

**Outcomes:** CS 1, CS 6, 5870-1

# 11 [SW 1.4.17]

*One-dimensional farthest pair*. Write an algorithm that, given an array of $N$ numbers, finds a *farthest pair*: two values whose difference is no smaller than the difference of any other pair (in absolute value). The running time of your algorithm must be linear in the worst case. Bonus points if you can do exactly $\lfloor \frac{3N}{2} - \frac{3}{2} \rfloor$ comparisons.

**Answer:**

**Algorithm 5** One-dimensional Farthest pair

```
minVal ← A[0]
maxVal ← A[0]
for i ← 1 to n − 1 do
    if A[i] < minVal then
        minVal ← A[i]
    end if
    if A[i] > maxVal then
        maxVal ← A[i]
    end if
end for
return minVal, maxVal
```

**Outcomes:** CS 2, 5870-1, 5870-2

# 12 [GT A-1.12]

Given an array $A$ of $n-2$ unique integers in the range from 1 to $n$, describe an $O(n)$-time algorithm for finding the two integers in the range from 1 to $n$ that are not in $A$. You may use only $O(1)$ space in addition to the space used by $A$.

**Answer:**

**Algorithm 6** Array A of size n - 2 (with elements from 1 to n, missing two numbers)

$totalSum \leftarrow \frac{n(n+1)}{2}$
$totalSumSq \leftarrow \frac{n(n+1)(2n+1)}{6}$
$arraySum \leftarrow 0$
$arraySumSq \leftarrow 0$
**for** $i \leftarrow 0$ to $n-3$ **do**
    $arraySum \leftarrow arraySum + A[i]$
    $arraySumSq \leftarrow arraySumSq + A[i]^2$
**end for**
$sumDiff \leftarrow totalSum - arraySum$ ▷ $x + y$
$sumSqDiff \leftarrow totalSumSq - arraySumSq$ ▷ $x^2 + y^2$
$product \leftarrow \frac{sumDiff^2 - sumSqDiff}{2}$ ▷ $xy$ ▷ Solve quadratic: $t^2 - sumDiff \cdot t + product = 0$
$d \leftarrow sumDiff^2 - 4 \cdot product$
$x \leftarrow \frac{sumDiff + \sqrt{d}}{2}$
$y \leftarrow \frac{sumDiff - \sqrt{d}}{2}$
**return** $(x, y)$

**Outcomes:** CS 1, CS 2, 5870-1