CS213/293 Data Structure and Algorithms 2025

Lecture 1: आपको "डेटा स्ट्रक्चर" की पढ़ाई क्यों करनी चाहिए?

(Why should you study "data structures?")

Instructor: Ashutosh Gupta

IITB India

Compile date: August 9, 2025

अगला कोर्स कोडिंग के लिए (Next course in programming)

क्या प्रोग्रामर बनने के लिए CS101 और SSL पूरे नहीं हैं? (Are CS101 and SSL not enough to be a programmer?)

CS101 में आपने चलना सीखा।
(In CS101, you learned to walk.)

इस कोर्स में आप नाचना करना सीखेंगे।
(In this course, you will learn to dance.)

डेटा क्या है? (What is data?)

चीज़ें डेटा नहीं हैं, लेकिन उनके बारे में जानकारी डेटा हैं।
(Things are not data, but information about them is data.)

Example 1.1

लोगों की आयु, पेड़ों की ऊंचाई, स्टॉकों की कीमत, और लाइकों की संख्या.
(Age of people, height of trees, price of stocks, and number of likes.)

डेटा बहुत सारा है! (Data is big!)

हम डेटा के सागर में रह रहे हैं! (We are living in the age of big data!)

*Image is from the Internet.

Exercise 1.1

1. व्हाट्सएप स्टेटस के लिए भेजे गए संदेशों की संख्या का अनुमान लगाएं।
   (Estimate the number of messages exchanged for Whatsapp status.)
2. चैटजीपीटी को ट्रेन करने के लिए कितने डेटा का उपयोग किया गया है?
   (How much text data was used to train ChatGPT?)

हमें इस डाटा पर ही काम करना है (We need to work on data)

हम अपनी समस्याओं को हल करने के लिए डेटा को process करते हैं। (We process data to solve our problems.)

Example 1.2

1. मौसम का पूर्वानुमान लगाएं (Predict the weather)
2. वेबपेज खोजें (Find a webpage)
3. फिंगरप्रिंट पहचानें (Recognize fingerprint)

अव्यवस्थित डेटा को process करने में बहुत समय लगेगा। (Disorganized data will need a lot of time to process.)

Exercise 1.2

किसी ऐरे में एक एलिमेंट ढूंढने के लिए हमें कितना समय चाहिए? (How much time do we need to find an element in an array?)

समस्या (Problem)

Definition 1.1

एक समस्या एक input specification और एक output specification की एक जोड़ी है।
(A problem is a pair of an input specification and an output specification.)

Example 1.3

search की समस्या के निम्नलिखित specifications हैं। (The problem of search consists of the following specifications)

- Input specification: एलिमेंटस की एक ऐरे S और एक एलिमेंट e (an array S of elements and an element e)
- Output specification: S में e की स्थिति यदि वह मौजूद है। यदि e नहीं मिलता, तो -1 लौटाएँ।
  (position of e in S if it exists. If it is not found, return -1.)

Output specification input specification के variables का जिक्र कर सकते हैं
(Output specifications refer to the variables in the input specifications)

Exercise 1.3

Specification के अनुसार, यदि e, S में कई बार आता है तो क्या होना चाहिए?
(According to the specification, what should happen if e occurs multiple times in S?)

एल्गोरिदम (Algorithms)

Definition 1.2

एक एल्गोरिदम किसी दी गई समस्या को हल करता है। (An algorithm solves a given problem.)

- Input $\in$ Input specifications
- Output $\in$ Output specifications

$$\text{Input} \longrightarrow \boxed{\text{एल्गोरिदम}} \longrightarrow \text{Output}$$

नोट: किसी समस्या को हल करने के लिए कई एल्गोरिदम हो सकती हैं। (Note: There can be many algorithms to solve a problem.)

Exercise 1.4

1. वास्तव में एल्गोरिदम है क्या? (What truly is an algorithm?)
2. एक एल्गोरिदम एक प्रोग्राम से कैसे अलग है? (How an algorithm is different from a program?)

Commentary: एक एल्गोरिदम एक step-by-step प्रक्रिया होती है जो प्रत्येक कदम में थोड़ी मात्रा का डाटा process करती है और अंततः आउटपुट की गणना करती है। एल्गोरिदम की औपचारिक परिभाषा आपको CS310 में प्रस्तुत की जाएगी। एक एल्गोरिदम की सटीक परिभाषा देने के लिए एलन ट्यूरिंग की प्रतिभा की आवश्यकता पड़ी थी।

उदाहरण: search के लिए एक एल्गोरिदम (Example: an algorithm for search)

Example 1.4

```
int search( int* S, int n, int e) {
  // n ऐरे S की लंबाई है (n is the length of the array S)
  // हम S में एलिमेंट e की तलाश कर रहे हैं (We are looking for element e in S)
  for( int i=0; i < n; i++ ) {
    if( S[i] == e ) {
      return i;
    }
  }
  return -1; // नहीं मिला (Not found)
}
```

Exercise 1.5

यदि e, S में नहीं है तो इस एल्गोरिदम का चलने का समय क्या है? (What is the running time of the above algorithm if e is not in S?)

Commentary: उत्तर: हम मेमोरी एक्सेस, गणितीय operations (तुलनाएं सहित), असाइनमेंट्स, और जम्प्स की गिनती करते हैं। प्रोग्राम में लूप n बार इटरेट होगा। प्रत्येक इटरेशन में, एक मेमोरी एक्सेस होगा `S[i]`, तीन गणितीय operations होंगे `i<n`, `S[i] == e` और `i++`, और दो जम्प्स होंगे। इनिशियलाइज़ेशन में, एक असाइनमेंट होगा `i=0` लूप के बाहर निकलने के लिए, एक और तुलना और जम्प होगा। $Time = nT_{Read} + (3n+2)T_{Arith} + (2n+1)T_{jump} + T_{return}$ इस प्रोग्राम को `https://godbolt.org/` को दें और असेंबली देखें। जांचें कि क्या ऊपरी विश्लेषण विश्वसनीय है!

डेटा को स्ट्रक्चर की आवश्यकता है (Data needs structure)

डेटा को सामान के ढेर के रूप में रखने से काम नहीं चलेगा। हमें स्ट्रक्चर की आवश्यकता है।
(Storing data as a pile of stuff, will not work. We need structure.)

Example 1.5

हम आईआईटी बॉम्बे अस्पताल में फ़ाइलें कैसे रखते हैं? (How do we store data at IIT Bombay Hospital?)

स्ट्रक्चरड डेटा हमें समस्याओं को तेज़ी से हल करने में मदद करता है

(Structured data helps us solve problems faster)

हम अपनी समस्याओं को हल करने के लिए कुशल एल्गोरिदम डिजाइन करने के लिए स्ट्रक्चर का लाभ उठा सकते हैं।
(We can exploit the structure to design efficient algorithms to solve our problems.)

इस कोर्स का लक्ष्य! (The goal of this course!)

उदाहरण: सुव्यवस्थित डेटा पर search I (Example: search on well-structured data I)

Example 1.6

आइए हम देखें search की समस्या को, जिसके निम्नलिखित specifications हैं।

(Let us consider the problem of search consisting of the following specifications)

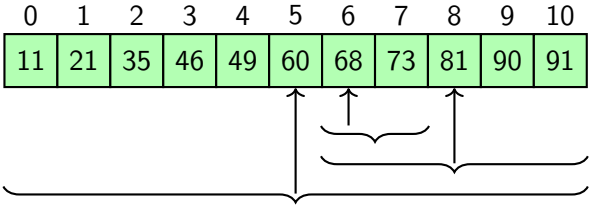
- Input specification: एक न घटने वाली ऐरे S और एक एलिमेंट e (a non-decreasing array S and an element e)
- Output specification: S में e की स्थिति यदि वह मौजूद है। यदि e नहीं मिलता, तो -1 लौटाएँ।
  (position of e in S if it exists. If it is not found, return -1.)

उदाहरण: सुव्यवस्थित डेटा पर search II (Example: search on well-structured data I)

आइए देखें कि हम सुव्यवस्थित डेटा का कैसे इस्तेमाल कर सकते हैं! (Let us see how we can exploit the structured data!)

आइए हम निम्नलिखित ऐरे में 68 की search का प्रयास करते हैं। (Let us try to search 68 in the following array.)

- ऐरे के बीच का एलिमेंट देखें।
  (Look at the middle point of the array.)
- चूंकि बीच के एलिमेंट का मान 68 से कम है, हम केवल ऊपरी आधे भाग में search करेंगें।
  (Since the value at the middle point is less than 68, we search only in the upper half.)
- हमने अपनी search space को आधा कर दिया है।
  (We have halved our search space.)
- हम recursively space को आधा करते हैं।
  (We recursively half the space.)



A better search

Example 1.7

```
int BinarySearch(int* S, int n, int e){
  // S एक न घटने वाली ऐरे है। (S is a sorted array)
  int first = 0, last = n;
  int mid = (first + last) / 2;
  while (first < last) {
    if (S[mid] == e) return mid;
    if (S[mid] > e) {
      last = mid;
    } else {
      first = mid + 1;
    }
    mid = (first + last) / 2;
  }
  return -1;
}
```

Commentary: उत्तर: वहाँ k iterations होंगी। प्रत्येक iteration में, फंक्शन एक ही पथ का पालन करेगा। प्रत्येक iteration में, निम्नलिखित होगा

- एक मेमोरी एक्सेस `S[mid]`,(क्यों सिर्फ एक?)
- पांच गणितीय operations `first < last`, `S[mid] == e`, `S[mid] > e`, `first+last`, और `../2`,
- एक असाइनमेंट `last = mid`, क्यों?
- दो ifs और एक लूप समाप्ति के कारण तीन कूद,

लूप निकास के लिए, लूप हेड पर एक अतिरिक्त तुलना और एक कूद होगी। आरंभिकरण खंड में, हमारे पास दो असाइनमेंट्स और दो गणितीय operations हैं।

$Time = kT_{Read} + (6k+5)T_{Arith} + (3k+1)T_{jump} + T_{return}$

Exercise 1.6

चलिए माना $n = 2^{k-1}$, इस एल्गोरिदम को चलाने में कितना समय लगेगा यदि `S[0] > e` है?

(Let $n = 2^{k-1}$. How much time will it take to run the algorithm if `S[0] > e`?)

Topic 1.1

बिग-ओ संकेतन (Big-O notation)

एक एल्गोरिदम को कितना संसाधन चाहिए? (How much resource does an algorithm need?)

एक समस्या को हल करने के लिए कई एल्गोरिदम हो सकते हैं। (There can be many algorithms to solve a problem.)

कुछ अच्छे हैं और कुछ बुरे हैं। (Some are good and some are bad.)

अच्छे एल्गोरिदम दो मायनों में कुशल हो सकते है (Good algorithms are efficient in)

- समय और (time and)
- स्पेस. (space.)

हमारा समय मापने का तरीका बोझिल और मशीन-निर्भर है। (Our method of measuring time is cumbersome and machine-dependent.)

हमें अनुमानित गिनती की आवश्यकता है जो मशीन-स्वतंत्र है। (We need approximate counting that is machine-independent.)

Commentary: कभी-कभी समय और स्पेस की जरूरत के बीच समझौता करना पड़ता है। उदाहरण के लिए, linear search केवल एक अतिरिक्त integer की आवश्यकता होती थी, लेकिन binary search ने तीन अतिरिक्त integers का उपयोग किया। दो integers के बीच अंतर एक मामूली मुद्दा हो सकता है, लेकिन यह समझौते की जरूरत को दर्शाता है।

इनपुट का आकार (Input size)

एक एल्गोरिदम का चलने का समय अलग-अलग इनपुट्स के लिए अलग हो सकता है।
(An algorithm may have different running times for different inputs.)

हमें एल्गोरिदम्स की तुलना करने के बारे में कैसे सोचना चाहिए? (How should we think about comparing algorithms?)

इनपुट के महत्वपूर्ण पैरामीटर का इस्तेमाल करके इनपुट के लगभग आकर को मायने देते हैं।
(We define the rough size of the input, usually in terms of important parameters of input.)

Example 1.8

search की समस्या में, हम कहते हैं कि ऐरे में एलिमेंट्स की संख्या इनपुट का आकार है।
(In the problem of search, we say that the number of elements in the array is the input size.)

कृपया ध्यान दें कि व्यक्तिगत एलिमेंट्स के आकार का इस्तेमाल नहीं किया गया है।
(Please note that the size of individual elements is not considered.) (Why?)

Commentary: वास्तव में हमें इनपुट की बिट्स की संख्या को काउंट करना चाहिए, जो की मुश्किल है. search की समस्या में हम मानते हैं की सरे एलिमेंट्स 2^{32} से छोटे हैं. इसलिए एलिमेंट्स का साइज 32 बिट्स से छोटा होगा.

Best/Average/Worst case

एक आकार के इनपुट के लिए, हम निम्नलिखित परिस्तिथियाँ सोंच सकते हैं।
(For a given size of inputs, we may further make the following distinction.)

1. Best case: किसी इनपुट के लिए सबसे कम समय लेना (Shortest running time for some input.)
2. Worst case: किसी इनपुट के लिए सबसे ज्यादा समय लेना (Worst running time for some input.)
3. Average case: एक दिए गए आकार वाले सभी इनपुट्स पर औसत चलने का समय
   (Average running time on all the inputs of the given size.)

Exercise 1.7

हम किसी भी एल्गोरिदम को कैसे बदल सकते हैं ताकि उसका best case समय अच्छा हो?
(How can we modify almost any algorithm to have a good best-case running time?)

Example: Best/Average/Worst case

Example 1.9

```
int BinarySearch(int* S, int n, int e){
  // S एक न घटने वाली ऐरे है। (S is a sorted array)
  int first = 0, last = n;
  int mid = (first + last) / 2;
  while (first < last) {
    if (S[mid] == e) return mid;
    if (S[mid] > e) {
      last = mid;
    } else {
      first = mid + 1;
    }
    mid = (first + last) / 2;
  }
  return -1;
}
```

BinarySearch में, मान लें कि $n = 2^{k-1}$
(In BinarySearch, let $n = 2^{k-1}$.)

1. Best case: $e == S[n/2]$
   $T_{Read} + 6T_{Arith} + T_{return}$,
2. Worst case: $e \notin S$
   हमने सबसे ज्यादा समय लेने वाला इनपुट देखा है। (We have seen the worst case.)
3. average case समय लगभग worst case के बराबर होगा क्योंकि अधिकांश समय लूप k बार चलेगा। (The average case is roughly equal to the worst case because most often the loop will iterate k times. (Why?))

Commentary: average case का विश्लेषण करना आमतौर पर मुश्किल होता है, जो इनपुट्स के वितरण पर निर्भर करता है। कुछ महत्वपूर्ण एल्गोरिदमों के लिए, हम average case का विस्तृत विश्लेषण करेंगे।

Asymptotic और मशीन-स्वतंत्र व्यवहार (Asymptotic and machine-independent behavior)

छोटे इनपुट्स के लिए, एक एल्गोरिदम बेहतर समय के लिए एक शॉर्टकट का उपयोग कर सकता है।
(For short inputs, an algorithm may use a shortcut for better running time.)

ऐसी झूठी तुलनाओं से बचने के लिए, हम अल्गोरिदम्स के व्यवहार को सीमा में देखते हैं।
(To avoid such false comparisons, we look at the behavior of the algorithms in limit.)

हार्डवेयर-निर्भर विवरणों को नजरअंदाज करें (Ignore hardware-specific details)

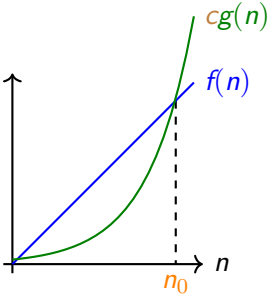
- Rounding numbers: $100000000000001 \approx 100000000000000$
- Ignore coefficients: $3kT_{Arith} \approx k$

बिग-ओ संकेतन: एक अनुमानित माप (Big-O notation: an approximate measure)

Definition 1.3

मान लें f और g फ़ंक्शन $\mathbb{N} \to \mathbb{N}$ हैं। $f(n) \in O(g(n))$ यदि c और n_0 ऐसे हैं कि
(Let f and g be functions $\mathbb{N} \to \mathbb{N}$. We say $f(n) \in O(g(n))$ if there are c and n_0 such that)

$$\text{सारे } n \geq n_0 \text{ के लिए} \qquad f(n) \leq cg(n).$$



- किसी समय के बाद, $cg(n)$ $f(n)$ पर हावी होगा (In limit, $cg(n)$ will dominate $f(n)$)
- हम कहते हैं $O(g(n))$ में $f(n)$ है (We say $f(n)$ is $O(g(n))$)

Exercise 1.8

निम्नलिखित में से कौन से कथन सही हैं? (Which of the following are the true statements?)

- $5n + 8 \in O(n)$
- $5n + 8 \in O(n^2)$
- $5n^2 + 8 \in O(n)$
- $n^2 + n \in O(n^2)$
- $5000000000000000000000000n^2 \in O(n^2)$
- $50n^2 logn + 60n^2 \in O(n^2 logn)$

उदाहरण: BinarySearch के worst case का बिग-ओ (Example: Big-O of the worst case of BinarySearch)

Example 1.10

BinarySearch में, मान लें कि $n = 2^{k-1}$। (In BinarySearch, let $n = 2^{k-1}$.)

worst case समय $kT_{Read} + (6k+5)T_{Arith} + (3k+1)T_{jump} + T_{return}$ है।

इसलिए, $kT_{Read} + (6k+5)T_{Arith} + (3k+1)T_{jump} + T_{return} \in O(k)$.

चूंकि $k = log\ n + 1$, इसलिए $k \in O(log\ n)$.

इसलिए, BinarySearch का worst case समय $O(log\ n)$ है।
(Therefore, the worst-case running time of BinarySearch is $O(log\ n)$.)

हम यह भी कह सकते हैं कि BinarySearch $O(log\ n)$ है।
(We may also say BinarySearch is $O(log\ n)$.)

Exercise 1.9

साबित करें कि $f \in O(g)$ और $g \in O(h)$, तो $f \in O(h)$।
(Prove that $f \in O(g)$ and $g \in O(h)$, then $f \in O(h)$.)

बिग ओ क्या कहता है? (What does Big O say?)

बिग ओ प्रोग्राम द्वारा executed operations की अनुमानित संख्या को व्यक्त करता है जो इनपुट आकार के फ़ंक्शन के रूप में लिखा जाता है। (Big O expresses the approximate number of operations executed by the program as a function of input size.)

एल्गोरिदम्स का क्रम (Hierarchy of algorithms)

- $O(log\ n)$ एल्गोरिदम $O(n)$ से बेहतर है। ($O(log\ n)$ algorithm is better than $O(n)$)
- हम कहते हैं: $O(log\ n) < O(n) < O(n^2) < O(2^n)$

बड़े constants को छुपा सकता है!! (May hide large constants!!)

Exercise 1.10

कृपया फ़ंक्शन्स के ऊपर $<$ की औपचारिक परिभाषा दें। (Give formal definition of $<$ over functions.)

एक एल्गोरिदम की कॉम्प्लेक्सिटी (Complexity of an algorithm)

Definition 1.4

एक एल्गोरिदम का worst-case समय एल्गोरिदम की कॉम्प्लेक्सिटी होती है।
(The worst-case running time of an algorithm is the complexity of the algorithm.)

हम average-case कॉम्प्लेक्सिटी को निम्नलिखित रूप में परिभाषित कर सकते हैं।
(We may also define average-case complexity as follows.)

Definition 1.5

एक एल्गोरिदम का average-case समय एल्गोरिदम की average-case कॉम्प्लेक्सिटी होती है।
(The average-case running time of an algorithm is the average-case complexity of the algorithm.)

Example 1.11

`BinarySearch` की कॉम्प्लेक्सिटी $O(\log\ n)$ है। (The complexity of `BinarySearch` is $O(\log\ n)$.)

Commentary: कॉम्प्लेक्सिटी के बयान में, जैसा कि हमने पहले चर्चा की थी, कई सूक्ष्म assumptions होती हैं। हमें इन assumptions का हमेशा ध्यान रखना चाहिए, जब हम कॉम्प्लेक्सिटी विश्लेषण करते हैं। कभी-कभी assumptions कॉम्प्लेक्सिटी विश्लेषण की उपयोगिता को सीमित कर देती हैं।

एक समस्या की कॉम्प्लेक्सिटी (Complexity of a problem)

समस्या की कॉम्प्लेक्सिटी उस समस्या के लिए सर्वश्रेष्ठ ज्ञात एल्गोरिदम की कॉम्प्लेक्सिटी होती है।
(The complexity of a problem is the complexity of the best-known algorithm for the problem.)

Exercise 1.11

इस समस्या की कॉम्प्लेक्सिटी क्या है? (What is the complexity of the following problem?)

- sorting an array $O(n^2)$ ✗
- matrix multiplication $O(n^3)$ ✗

सर्वश्रेष्ठ एल्गोरिदम अभी तक नहीं ज्ञात है।
(Best algorithm is still not known)

Exercise 1.12

उपरोक्त समस्याओं के लिए सबसे अच्छी तरह से ज्ञात कॉम्प्लेक्सिटी क्या है?
(What is the best-known complexity for the above problems?)

Commentary: matrix multiplication एल्गोरिदम में नवीनतम विकासों पर चर्चा: https://en.wikipedia.org/wiki/Computational_complexity_of_matrix_multiplication

कॉम्प्लेक्सिटी वर्गों के नाम (Names of complexity classes)

- Constant: $O(1)$
- Logarithmic: $O(logn)$
- Linear: $O(n)$
- Quadratic: $O(n^2)$
- Polynomial : $O(n^k)$ for some given k
- Exponential : $O(2^n)$

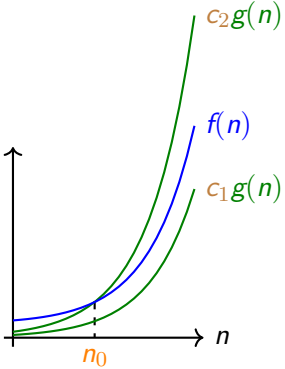
Θ-Notation

Definition 1.6 (Tight bound)

मान लें f और g फ़ंक्शन $\mathbb{N} \to \mathbb{N}$ हैं। $f(n) \in \Theta(g(n))$ यदि $c_1 > 0$, $c_2 > 0$, और n_0 ऐसे हैं कि

(Let f and g be functions $\mathbb{N} \to \mathbb{N}$. We say $f(n) \in \Theta(g(n))$ if there are $c_1 > 0$, $c_2 > 0$, and n_0 such that)

$$\text{सारे } n \geq n_0 \text{ के लिए} \qquad c_1 g(n) \leq f(n) \leq c_2 g(n)$$



उपरोक्त परिभाषा के अधिक संस्करण हैं। कृपया अंत में देखें। (There are more variations of the above definition. Please look at the end.)

Exercise 1.13

a. क्या `BinarySearch` की कॉम्प्लेक्सिटी $\Theta(\log n)$ है? (Does the complexity of `BinarySearch` belong to $\Theta(\log n)$?)

b. अगर हां, तो `BinarySearch` पर ऊपरी परिभाषा के अनुप्रयोग के लिए c_1, c_2, और n_0 दें।

(If yes, give c_1, c_2, and n_0 for the application of the above definition on `BinarySearch`.)

Topic 1.2

Tutorial Problems

Problem: Compute the exact running time of insertion sort.

Exercise 1.14

The following is the code for insertion sort. Compute the exact worst-case running time of the code in terms of n and the cost of doing various machine operations.

```
for( int j = 1; j < n; j++ ) {
  int key = A[j];
  int i = j-1;
  while( i >= 0 ) {
    if( A[i] > key ) {
      A[i+1] = A[i];
    }else{
      break;
    }
    i--;
  }
  A[i+1] = key;
}
```

Problem: additions and multiplication

Exercise 1.15

What is the time complexity of binary addition and multiplication? How much time does it take to do unary addition?

Problem: hierarchy of complexity

Exercise 1.16

Given $f(n) = a_0 n^0 + ... + a_d n^d$ and $g(n) = b_0 n^0 + ... + b_e n^e$ with $d > e$ and $a_d > 0$ (Why?), show that $f(n) \notin O(g(n))$.

Topic 1.3

Problems

True or False

Exercise 1.17

Mark the following statements True / False. Also provide justification.

1. For any function $f : \mathbb{N} \to \mathbb{N}$, $O(f) \subseteq \Omega(f)$.
2. For a fixed array of size 2^k for integer k, the binary search always takes the same amount of time in the case of an unsuccessful search.

Order of functions

Exercise 1.18

- If $f(n) \leq F(n)$ and $G(n) \geq g(n)$ (in order sense) then show that $\frac{f(n)}{G(n)} \leq \frac{F(n)}{g(n)}$.
- Is $f(n)$ the same order as $f(n)|sin(n)|$?

Exercise: an important complexity class!

Exercise 1.19

Prove that $O(\log(n!)) = O(n \log n)$. Hint: Stirling's approximation

Exercise: egg drop problem**

Exercise 1.20

In the dead of night, a master jewel thief is plotting the heist of a lifetime-stealing the most valuable Faberge Egg from a towering 100-story museum. Each floor of the building has an identical egg, but the higher the floor, the more valuable the egg becomes. However, there's a catch. The thief can steal only one egg and she knows that the most valuable egg at the top may not survive a drop from such a great height. To avoid smashing her prized loot, she must identify the highest floor from which an egg can be dropped without breaking. Armed with two replica eggs from the museum's gift shop-perfectly identical but utterly worthless-the thief devises a plan. These two eggs will be her test subjects, sacrificed in the pursuit of the perfect drop. But time is of the essence, and the thief can not afford to be caught by the museum guards. She needs to figure out the minimum number of test drops required to guarantee finding the highest safe floor. Once an egg is broken, it's gone for good-no replacements, no second chances. She cannot use any other method to determine the sturdiness of the eggs.

a. Give an algorithm for the thief to determine, with the least number of drops in the worst case, the highest floor from which an egg can be safely dropped without breaking. (Quiz 2024)

b. Give an algorithm for the best average case, assuming that the probability of the highest safe floor is uniformly distributed.

c. Prove optimality of your algorithm.***

Commentary: https://www.youtube.com/watch?v=NGtt7GJ1uiM

Identities for Big-O (Midsem 2024)

Definition 1.7

Let A and B be subsets of $\mathfrak{p}(\mathbb{N} \to \mathbb{N})$. $A + B = \{f + g | f \in A \wedge g \in B\}$.

Exercise 1.21

Prove/Disprove the following:

- $O(f)g \subseteq O(fg)$
- $O(f) + O(g) \subseteq O(f + g)$

Exercise 1.22

Can we give examples when the above subset relations are strict?

Topic 1.4

Extra slides: More on complexity

Ω notation

Definition 1.8 (Lower bound)

Let f and g be functions $\mathbb{N} \to \mathbb{N}$. We say $f(n) \in \Omega(g(n))$ if there are c and n_0 such that

$$cg(n) \leq f(n) \qquad \text{for all } n \geq n_0.$$

Small-o,ω notation

Definition 1.9 (Strict Upper bound)

Let f and g be functions $\mathbb{N} \to \mathbb{N}$. We say $f(n) \in o(g(n))$ if for each c, there is n_0 such that

$$f(n) \leq cg(n) \qquad \text{for all } n \geq n_0.$$

Definition 1.10 (Strict Lower bound)

Let f and g be functions $\mathbb{N} \to \mathbb{N}$. We say $f(n) \in \omega(g(n))$ if for each c, there is n_0 such that

$$cg(n) \leq f(n) \qquad \text{for all } n \geq n_0.$$

Exercise 1.23

a. Prove that $f \in o(g)$ implies $f \in O(g)$.
b. Show that $f \in O(g)$ does not imply $f \in o(g)$.

Size of functions

We can define a partial order over functions using the above notations

- $f(n) \in O(g(n))$ implies $f(n) \leq g(n)$
- $f(n) \in o(g(n))$ implies $f(n) < g(n)$
- $f(n) \in \Omega(g(n))$ implies $f(n) \geq g(n)$
- $f(n) \in \omega(g(n))$ implies $f(n) > g(n)$
- $f(n) \in \Theta(g(n))$ implies $f(n) = g(n)$

Exercise 1.24

Show that the partial order is well-defined.

Commentary: Why do we need to prove that the definition is well-defined?

Topic 1.5

Extra slides: Binary search in recursive representation!

Search for ordered keys

If keys are stored in order, then we use the binary search that we have discussed in lecture 1.

Algorithm 1.1: BinarySearch(A, key, low, high)

```
if low > high then
    return -1
mid := (low+high)/2;
if A[mid] == key then
    return mid
if key < A[mid] then
    return BinarySearch(A,key, low, mid-1)
return BinarySearch(A,key, mid+1, high)
```

Exercise 1.25

We earlier saw the iterative version of the Binary search. Can we write any recursive algorithm as iterative algorithm?

End of Lecture 1