ECS 122A: Algorithm Design and Analysis

Week 2 Discussion

Ji Wang

Fall 2020

A bit about logistics

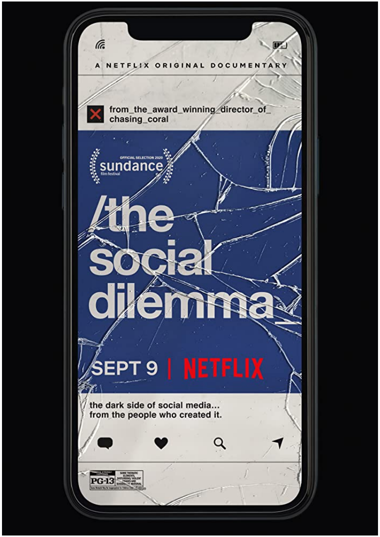
Discussion Schedule:

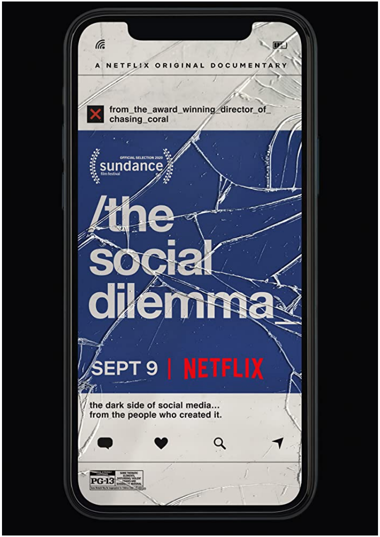
Ji Wang → Christopher Peterson → Terry Guan

- We will rotate to lead the discussion, with the exception of Veteran Day (Wed) and the entire Thanksgiving week.
- The first discussion of the week will be live and recorded. The rest will be in the form of live Q&A or extra office hours.
- Check Canvas homepage frequently for update on discussion notes and videos.

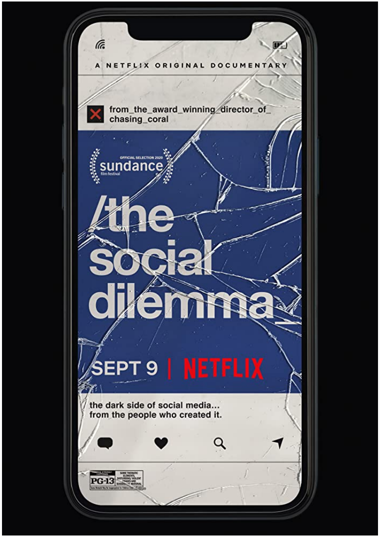
Office hours: Available every weekday
Ji: M 10:30am - 12:30pm, T 5 - 6pm, F 4 - 6pm

Contact us: For private questions, send direct message via Canvas.


A NETFLIX ORIGINAL DOCUMENTARY
from_the_award_winning_director_of_
chasing_coral
OFFICIAL SELECTION 2020
sundance
/the
social
dilemma_
SEPT 9 | NETFLIX
the dark side of social media...
from the people who created it.
PG-13



- Why these videos appear in my YouTube feed?
- Why the posts on Instagram not shown chronologically?
- Why my money gone easily when browsing Amazon?



- Why these videos appear in my YouTube feed?
- Why the posts on Instagram not shown chronologically?
- Why my money gone easily when browsing Amazon?

It's the **algorithms** that work under the hood!

Outline

- Proof technique: Mathematical Induction
- Data structure: Heap
- Design description: Pseudocode
- Analysis: Asymptotic Notation
- Example: Insertion Sort

Mathematical Induction: Approach

When applicable: Prove that a property $P(n)$ holds for every natural number n.

How it works:

1. Show that a property $P(n)$ holds for the base case, usually when $n = 0$ or 1.
2. Assume that $P(n)$ is true for $n = k$ where k is greater than the base case.
3. Prove that $P(n)$ is true for $n = k + 1$. In this step we will use the assumption above.
4. Then, by the principle of induction, we can conclude that $P(n)$ is true for every natural number that is greater than or equal to the base case.

Mathematical Induction: Example

Problem Statement: Prove the following statement $P(n)$ is true

$$\sum_{i=1}^{n} i^3 = (\frac{n(n+1)}{2})^2$$

Build a skeleton:

- Base Case:
- Inductive Hypothesis:
- Inductive Step:
- Thus, by the principle of induction, we can conclude that the statement above is true for every natural number n.

Mathematical Induction: Example

$$1^3 + 2^3 + \cdots + n^3 = (\frac{n(n+1)}{2})^2$$

- Base Case:
  Verify $P(n)$ is true when $n = 1$. $1^3 = (\frac{1 \cdot (1+1)}{2})^2$.

Mathematical Induction: Example

$$1^3 + 2^3 + \cdots + n^3 = (\frac{n(n+1)}{2})^2$$

- Base Case:
  Verify $P(n)$ is true when $n = 1$. $1^3 = (\frac{1 \cdot (1+1)}{2})^2$.
- Inductive Hypothesis:
  Assume $P(n)$ is true when $n = k$. Then, we have
  $1^3 + 2^3 + \cdots + k^3 = (\frac{k(k+1)}{2})^2$.

Mathematical Induction: Example

$$1^3 + 2^3 + \cdots + n^3 = (\frac{n(n+1)}{2})^2$$

- Inductive Step: We need to show $P(n)$ is true when $n = k+1$.

$$\begin{aligned}
1^3 + 2^3 + \cdots + (k+1)^3 &= 1^3 + 2^3 + \cdots + k^3 + (k+1)^3 \\
&= (\frac{k(k+1)}{2})^2 + (k+1)^3 \\
&= \frac{k^2(k+1)^2}{4} + \frac{4(k+1)^3}{4} \\
&= \frac{(k+1)^2(k^2 + 4(k+1))}{4} \\
&= \frac{(k+1)^2(k+2)^2}{4} \\
&= (\frac{(k+1)((k+1)+1)}{2})^2
\end{aligned}$$

Recap: Max-heap

Definition: Tree-based data structure which is an almost complete tree.

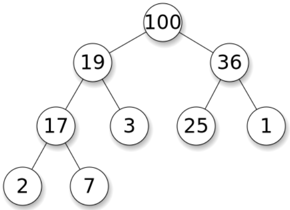
Property: For any given node, the key (value) of its parent node is greater than or equal to the that of itself.

Operations:

- find-max: find a maximum item of a max-heap. (peek)
- insert: add a new key to the heap. (push)
- extract-max: return the node of maximum value from a max-heap after removing it from the heap. (pop)
- increase-key or decrease-key: update a key within a max-heap.
- heapify: create a heap out of given array of elements.

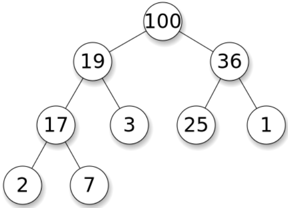
Applications: e.g. Prim's minimal-spanning-tree algorithm and Dijkstra's shortest-path algorithm

Max-heap: an example




```
push(10)
push(22)
pop()
update(7, 18)
```

Max-heap: an example




```
push(10)
push(22)
pop()
update(7, 18)
```

The time complexity of each operation is left to you.

How to describe the design of an algorithm

Objective: Understandable (clear, precise) and concise if possible.

[1]More pseudocode conventions can be found in textbook [pp.20-22]

How to describe the design of an algorithm

Objective: Understandable (clear, precise) and concise if possible.

1. in English
   - First (usually) points out what strategy/method used, e.g. divide-and-conquer, binary search.
   - Separate paragraphs if necessary, e.g. branches.
   - Bullet-point format is also a good practice.

[1] More pseudocode conventions can be found in textbook [pp.20-22]

How to describe the design of an algorithm

Objective: Understandable (clear, precise) and concise if possible.

1. in English
   - First (usually) points out what strategy/method used, e.g. divide-and-conquer, binary search.
   - Separate paragraphs if necessary, e.g. branches.
   - Bullet-point format is also a good practice.
2. in Pseudocode [1]
   - Indentation indicates block structure, similar to Python.
   - Indices (often) start from 1, unlike most languages.
   - Use comments as needed. (e.g. variable usage, function of a block)
   - Pass parameters to a procedure *by value*.

[1] More pseudocode conventions can be found in textbook [pp.20-22]

How to describe the design of an algorithm

Objective: Understandable (clear, precise) and concise if possible.

1. in English
   - First (usually) points out what strategy/method used, e.g. divide-and-conquer, binary search.
   - Separate paragraphs if necessary, e.g. branches.
   - Bullet-point format is also a good practice.
2. in Pseudocode [1]
   - Indentation indicates block structure, similar to Python.
   - Indices (often) start from 1, unlike most languages.
   - Use comments as needed. (e.g. variable usage, function of a block)
   - Pass parameters to a procedure *by value*.
3. in "real" code: whatever language you favor

[1] More pseudocode conventions can be found in textbook [pp.20-22]

How to Analyze an algorithm: Asymptotic notation

Analyzing an algorithm means predicting the resources that the algorithm needs. The primary concern in this course is to measure computational time, a.k.a. running time.

How to Analyze an algorithm: Asymptotic notation

Analyzing an algorithm means predicting the resources that the algorithm needs. The primary concern in this course is to measure computational time, a.k.a. running time.

We need a definition to evaluate the order of growth to:

- characterize the efficiency of the algorithm when input sizes are large enough.
- compare the performance with other alternative algorithms.

Thus, we:

- study a way to describe the growth of functions in the limit.
- focus on what's important (leading factor) by ignoring lower-order terms and constant factors.

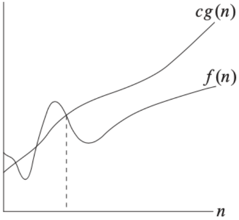
Asymptotic notation: Big O

Definition: $g(n)$ is an asymptotic upper bound for $f(n)$, denoted by

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

if there exist constants c and n_0 such that:

$$0 \leq f(n) \leq cg(n) \quad \text{for } n \geq n_0$$



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

[2]See auxiliary notes for reviews on frequent used functions

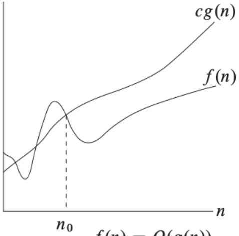
Asymptotic notation: Big O

Definition: $g(n)$ is an asymptotic upper bound for $f(n)$, denoted by

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

if there exist constants c and n_0 such that:

$$0 \leq f(n) \leq cg(n) \qquad \text{for } n \geq n_0$$



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

Example: Show $\lg n$ is $O(\ln n)$

[2] See auxiliary notes for reviews on frequent used functions

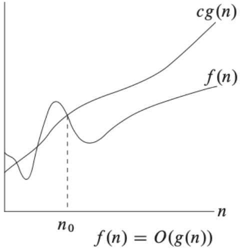
Asymptotic notation: Big O

Definition: $g(n)$ is an asymptotic upper bound for $f(n)$, denoted by

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

if there exist constants c and n_0 such that:

$$0 \leq f(n) \leq cg(n) \quad \text{for } n \geq n_0$$



Example: Show $\lg n$ is $O(\ln n)$

$$\lg n = \log_2 n = \frac{\log_e n}{\log_e 2} = \frac{1}{\log_e 2} \ln n$$
$$\leq 2 \ln n \quad \text{for } n \geq 1$$

it is true for $c = 2$ and $n_0 = 1$. [2]

[2] See auxiliary notes for reviews on frequent used functions

Example: Insertion Sort

Problem Statement: Given a list of numbers, sort them in non-decreasing order.

Example: Insertion Sort

Problem Statement: Given a list of numbers, sort them in non-decreasing order.

Algorithm: We take an *incremental* approach. Split the list into sorted part and unsorted part. Insert the head of unsorted one into the appropriate position in the sorted one. Repeat until no unsorted part remaining.

Example: 8 3 2 7 4 1 6

$\boxed{8}$ 3 2 7 4 1 6

$\boxed{8\ 3}$ 2 7 4 1 6

$\boxed{3\ 8}$ 2 7 4 1 6

$\boxed{3\ 8\ 2}$ 7 4 1 6

$\boxed{2\ 3\ 8}$ 7 4 1 6

$\boxed{2\ 3\ 8\ 7}$ 4 1 6

$\boxed{2\ 3\ 7\ 8}$ 4 1 6

$\boxed{2\ 3\ 7\ 8\ 4}$ 1 6

$\boxed{2\ 3\ 4\ 7\ 8}$ 1 6

$\boxed{2\ 3\ 4\ 7\ 8\ 1}$ 6

$\boxed{1\ 2\ 3\ 4\ 7\ 8}$ 6

$\boxed{1\ 2\ 3\ 4\ 7\ 8\ 6}$

$\boxed{1\ 2\ 3\ 4\ 6\ 7\ 8}$

Insertion Sort: Pseudocode and Time Analysis

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert key into the sorted list A[1 ··· j − 1].
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

- Worst case: reversed order, e.g. 5 4 3 2 1.

$$\sum_{j=2}^{n} \sum_{i=1}^{j-1} = O(n^2)$$

Insertion Sort: Pseudocode and Time Analysis

```
INSERTION-SORT(A)

for j = 2 to A.length
    key = A[j]
    // Insert key into the sorted list A[1 ··· j − 1].
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

- Worst case: reversed order, e.g. 5 4 3 2 1.

$$\sum_{j=2}^{n}\sum_{i=1}^{j-1} = O(n^2)$$

- Best case: already sorted

$$\sum_{j=2}^{n} = O(n)$$