Big-O and Sorting

Lecture 17
Outline

• Importance of Algorithm Analysis
• Runtime
• Bubble Sort
• Insertion Sort
• Selection Sort
• Merge Sort
Importance of Algorithm Analysis (1/2)

- **Performance** of algorithm refers to how quickly it executes and how much memory it requires
  - performance matters when amount of data gets large!
  - can observe and analyze performance, then revise algorithm to improve its performance

- Algorithm analysis is crucial to computing and will be a central topic in CS0200!
Importance of Algorithm Analysis (2/2)

• Factors that affect performance
  o computing resources
  o language
  o implementation
  o size of data, denoted n
    ▪ number of elements to be sorted
    ▪ number of elements in `ArrayList` to iterate through
    ▪ much faster to search through list of CS15 students than list of Brown students

• This lecture: a brief introduction to Algorithm Analysis!
• Goal: to maximize efficiency and conserve resources
Outline

- Importance of Algorithm Analysis
- Runtime
- Bubble Sort
- Insertion Sort
- Selection Sort
- Merge Sort
Performance of Algorithms

● How fast will N! run relative to N?
  ○ N! will take exponentially longer as N increases
  ○ less difference with small N but we care about large inputs
  ○ one algorithm could take 2 seconds and another take 1 hour to accomplish the same task

● How fast will recursive Fibonacci(N) run relative to N?
  ○ proportional to $2^N$ – consider how fast this function grows with N

● How fast will Towers of Hanoi run relative to the number of disks?
  ○ also proportional to $2^N$ for N disks
Runtime (1/2)

- In analyzing an algorithm, **runtime** is the total number of times "the principal activity" of all steps in that algorithm is performed
  - varies with input and almost always grows with input size N – key is how fast it grows as a function of N

- In most of computer science, we focus on **worst case runtime**
  - easier to analyze and important for unforeseen inputs

- **Average case** is what will typically happen. **Best case** requires least amount of work and is the best situation you could have
  - average case is important; best case is interesting, but not insightful
Runtime (2/2)

- How to determine runtime?
  - inspect pseudocode and determine number of statements executed by algorithm as a function of input size
  - allows us to evaluate approximate speed of an algorithm independent of hardware or software environment
  - memory use may be even more important than runtime for embedded devices
**Elementary Operations**

- Algorithmic “time” is measured in numbers of **elementary operations**
  - math (+, -, *, /, max, min, log, sin, cos, abs, ...)
  - comparisons ( ==, >, <=, ...)
  - function (method) **calls** and value **returns** (body of the method is separate)
  - variable assignment
  - variable increment or decrement
  - array **allocation** (declaring an array) and array **access** (retrieving an array from memory)
  - creating a new object (careful, object’s constructor may have elementary ops too!)

- For purpose of algorithm analysis, assume each of these operations takes same time: **"1 operation"**
  - we are only interested in “asymptotic performance” for large data sets, i.e., as N grows large (small differences in performance don’t matter)
Example: Constant Runtime

```java
public int addition(int x, int y) {
    return x + y; // 2 operations
}
```

- 2 operations – 1 addition, 1 return statement
- How many operations are performed if this function were to add ten integers? Would it still be constant runtime?
Example: Linear Runtime

```java
//find max of a set of positive integers
public int maxElement(int[] a) {
    int max = 0; //assignment, 1 op
    for (int i=0; i<a.length; i++){//2 ops per iteration
        if (a[i] > max) { //2 ops per iteration
            max = a[i]; //2 ops per iteration, sometimes
        }
    }
    return max; //1 op
}
```

- Worst case varies proportional to the size of the input list: \(6N + 3\)
- How many operations if the array had 1,000 elements?
- We’ll run the `for` loop proportionally more times as the input list grows
- Runtime increase is proportional to \(N\), **linear**

Only the largest \(N\) expression **without constants** matters! \(6N+3\), \(4N\), \(300N\) are all **linear** in runtime. More about this on following slides!
Example: Quadratic Runtime

public void printPossibleSums(int[] a) {
    for (i = 0; i < a.length; i++) {
        // 2 ops per iteration
        for (j = 0; j < a.length; j++) {
            // 2 ops per iteration
            System.out.println(a[i] + a[j]); // 4 ops per iteration
        }
    }
}

- Requires about $8N^2$ operations (it is okay to approximate!)
- Number of operations executed grows quadratically!
- If one element added to list, element must be added with every other element in list
- Notice that linear runtime algorithm on previous slide had only one for loop, while this quadratic one has two nested for loops, a typical $N^2$ pattern
Big-O Notation – OrderOf()

- But how to abstract from implementation…?
- **Big O** notation
- **O(N)** implies runtime is linearly proportional to number of elements/inputs in the algorithm (constant ops per element)
  - \((N \text{ elements}) \times (\text{constant ops/element}) = N \text{ operations}\)
- **O(N^2)** implies each element is operated on \(N\) times
  - \((N \text{ elements}) \times (N \text{ operations/element}) = N^2 \text{ operations}\)
- Only consider “asymptotic behavior” i.e., when \(N >> 1\)
  - \(N\) is tiny when compared to \(N^2\) for \(N >> 1\)
- **O(1)** implies that runtime does not depend on number of inputs
  - runtime is the same regardless of how large/small input size is
Big-O Constants

- **Important**: Only the largest \( N \) expression *without constants* matters.
- We are not concerned about runtime with small numbers of data – we care about running operations on large amounts of inputs.
  - \( 3N^2 \) and \( 500N^2 \) are both \( O(N^2) \) because the larger the input, the less the “500” and the “3” will affect the total runtime.
  - \( N/2 \) is \( O(N) \)
  - \( 4N^2 + 2N \) is \( O(N^2) \)
- Useful sum for analysis:
  \[
  1 + 2 + 3 + \cdots + N = \sum_{k=1}^{N} k = \frac{N(N+1)}{2}, \text{ which is } O(N^2)
  \]
Social Security Database Example (1/3)

- Hundreds of millions of people in the US have a number associated to them
- If 100,000 people are named John Doe, each has an individual SSN
- If the government wants to look up information they have on John Doe, they use his SSN
Social Security Database Example (2/3)

- Say it takes $10^{-4}$ seconds to perform a constant set of operations on one SSN
  - running an algorithm on 5 SSNs will take $5 \times 10^{-4}$ seconds, and running an algorithm on 50 will only take $5 \times 10^{-3}$ seconds
  - both are incredibly fast, difference in runtime might not be noticeable by an interactive user
  - this changes with large amounts of data, i.e., the actual SS Database
Social Security Database Example (3/3)

- Say it takes $10^{-4}$ seconds to perform a constant set of operations on one SSN
  - to perform algorithm with $O(N)$ on 300 million people will take **8.3 hours**
  - $O(N^2)$ takes **285,000 years**

- With large amounts of data, differences between $O(N)$ and $O(N^2)$ are HUGE!
Graphical Perspective (1/2) – Linear Plot

- $f(N)$ on a small scale →
Graphical Perspective (2/2) – Log Plot

- $f(N)$ on a larger scale $\rightarrow$
- For 10 million items ($N = 10^7$)…
  - and $O(\log_{10}N)$ runtime, perform roughly 7 operations
  - and $O(N)$ runtime, perform roughly 10 million operations
  - and $O(N^2)$ runtime, perform roughly 100 trillion operations
- really try to stay sub-quadratic!!
TopHat Question (1/3)

What is the big-O runtime of this algorithm?

```java
public int sumArray(int[] array){
    int sum = 0;
    for (int i = 0; i < array.length; i++){
        sum = sum + array[i];
    }
    return sum;
}
```

A) O(N)  B) O(N²)  C) O(1)  D) O(2^N)
What is the big-O runtime of this algorithm?

Consider the getLetter (or equivalent) method from TicTacToe:

```java
public String getLetter(){
    return this.letter;
}
```

A) O(N)  B) O(N^2)  C) O(1)  D) O(2^N)
TopHat Question (3/3)

What is the big-O runtime of this algorithm?

```java
public int sumSquareArray(int[][][] a){
    int sum = 0;
    for (int i = 0; i < a.length; i++){
        for (int j = 0; j < a[0].length; j++){
            sum = sum + a[j][i];
        }
    }
    return sum;
}
```

A) O(N)  B) O(N²)  C) O(1)  D) O(2^N)
Outline

- Importance of Algorithm Analysis
- Runtime
  - Bubble Sort
  - Insertion Sort
  - Selection Sort
  - Merge Sort
Sorting

• We use runtime analysis to help choose the best algorithm to solve a problem

• Two common problems: **sorting** and **searching** through a list of objects

• This lecture we will analyze different **sorting** algorithms to find out which is fastest
Sorting – Social Security Numbers

● Consider an example where run-time influences your approach

● How would you sort every SSN in the Social Security Database in increasing order?

● Multiple known algorithms for sorting a list
  o these algorithms vary in their Big-O runtime
Bubble Sort (1/2)

- Iterate through sequence, comparing each element to its right neighbor
- Exchange adjacent elements if necessary; largest element “bubbles” to the right
- End up with sorted sub-array on the right. Each time we go through the list, need to switch at least one item fewer than before
Bubble Sort (2/2)

- Iterate through sequence, comparing each element to its right neighbor
- Exchange adjacent elements if necessary; largest element “bubbles” to the right
- End up with sorted sub-array on the right. Each time we go through the list, need to switch at least one item fewer than before
- $N$ is number of objects in the sequence

```java
int i = array.length;
boolean sorted = false;
while ((i > 1) && (!sorted)) {
    sorted = true;
    for(int j = 1; j < i; j++) {
        if (a[j-1] > a[j]) {
            int temp = a[j-1];
            a[j-1] = a[j];
            a[j] = temp;
            sorted = false;
        }
    }
    i--;
}
```
Bubble Sort - Runtime

### Worst-case analysis:

- **while** loop iterates N-1 times
- iteration \( i \) has \( 2 + 13(i - 1) \) operations

### Total:

\[
2 + N + 2(N-1) + 13[(N-1) + ... + 2 + 1] = 3N + 13N(N-1)/2 = 13N^2 + ... = O(N^2)
\]
Outline

- Importance of Algorithm Analysis
- Runtime
- Bubble Sort
- Insertion Sort
- Selection Sort
- Merge Sort
Insertion Sort (1/2)

- Like inserting a new card into a partially sorted hand by bubbling to the left in a sorted subarray
  - less brute force than bubble sort
- Add one element $a[i]$ at a time
- Find proper position, $j + 1$, to the left by shifting neighbors on the left ( $a[i-1], a[i-2], \ldots, a[j+1]$ ) to the right, until $a[j] < a[i]$
- Move $a[i]$ into vacated $a[j+1]$
- After iteration $i < a.length$, original $a[0] \ldots a[i]$ are in sorted order, but not necessarily in final position
Insertion Sort (2/2)

for (int i = 1; i < a.length; i++) {
    int toInsert = a[i];
    int j = i-1;
    while ((j >= 0) && (a[j] > toInsert)) {
        a[j+1] = a[j];
        j--;
    }
    a[j+1] = toInsert;
}
Insertion Sort - Runtime

```java
for (int i = 1; i < a.length; i++) {
    int toInsert = a[i];
    int j = i-1;
    while ((j >= 0) && (a[j] > toInsert)){
        a[j+1] = a[j];
        j--;
    }
    a[j+1] = toInsert;
}
```

- **while** loop inside our **for** loop
  - while loop calls on 1, 2, ..., N-1 operations
  - for loop calls the while loop N times
- **$O(N^2)$** because we have to call on a while loop with around N operations N different times
- Reminder: **constants do NOT** matter with Big-O!
Outline

• Importance of Algorithm Analysis
• Runtime
• Bubble Sort
• Insertion Sort
• Selection Sort
• Merge Sort
Selection Sort (1/2)

- Find smallest element and put it in a[0]
- Find 2\textsuperscript{nd} smallest element and put it in a[1], etc.
- Less data movement (no bubbling)
Selection Sort (2/2)

What we want to happen:

```java
int n = a.length;
for (int i = 0; i < n; i++) {
    find minimum element a[min] in subsequence a[i...n-1]
    swap a[min] and a[i]
}
```

```java
for (int i = 0; i < n-1; i++) {
    int min = i;
    for (int j = i + 1; j < n; j++) {
        if (a[j] < a[min]) {
            min = j;
        }
    }
    temp = a[min];
    a[min] = a[i];
    a[i] = temp;
}
```
Selection Sort - Runtime

- Most executed instructions are those in inner `for` loop

- Each instruction is executed \((N-1) + (N-2) + \ldots + 2 + 1\) times

- Time Complexity: \(O(N^2)\)

```java
for (int i = 0; i < n-1; i++) {
    int min = i;
    for (int j = i + 1; j < n; j++) {
        if (a[j] < a[min]) {
            min = j;
        }
    }
    temp = a[min];
    a[min] = a[i];
    a[i] = temp;
}
```
Comparison of Basic Sorting Algorithms

- Differences in **Best** and **Worst** case performance result from state (ordering) of input before sorting

- Selection Sort wins on data movement

- For small data, even the worst sort – Bubble (based on comparisons and movements) – is fine!

<table>
<thead>
<tr>
<th></th>
<th>Selection</th>
<th>Insertion</th>
<th>Bubble</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Comparisons</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Best</td>
<td>$n^2/2$</td>
<td>$n$</td>
<td>$n$</td>
</tr>
<tr>
<td>Average</td>
<td>$n^2/2$</td>
<td>$n^2/4$</td>
<td>$n^2/4$</td>
</tr>
<tr>
<td>Worst</td>
<td>$n^2/2$</td>
<td>$n^2/2$</td>
<td>$n^2/2$</td>
</tr>
<tr>
<td><strong>Movements</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Best</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Average</td>
<td>$n$</td>
<td>$n^2/4$</td>
<td>$n^2/2$</td>
</tr>
<tr>
<td>Worst</td>
<td>$n$</td>
<td>$n^2/2$</td>
<td>$n^2/2$</td>
</tr>
</tbody>
</table>
Outline

- Importance of Algorithm Analysis
- Runtime
- Bubble Sort
- Insertion Sort
- Selection Sort
- Merge Sort
Merge Sort
Recap: Recursion (1/2)

- Recursion is a way of solving problems by breaking them down into smaller sub-problems, and using results of sub-problems to find the answer.

- Example: You want to determine what row number you’re sitting in, but you can only get information by asking the people in front of you.
  - they also don’t know what row they’re in, and must ask people in front of them.
  - people in first row know that they’re row 1, since there is no row in front (base case).
  - they tell people behind them, who know that they’re 1 behind row 1, so they are row 2, etc.
  - this “unwinds” the recursion.
Recap: Recursion (2/2)

```java
public int findRowNumber(Row myRow) {
    if (myRow.getRowAhead() == null) { // base case!
        return 1;
    } else { // recursive case - ask the row in front
        int rowAheadNum = this.findRowNumber(myRow.getRowAhead());
        // my row number is one more than the row ahead’s number
        return rowAheadNum + 1;
    }
}
```
Recursion (Top Down) Merge Sort (1/7)

- Let's say you don't know how to sort n elements, but you have a friend who can sort any number less than n. How can you use the results to do your work? (similar to row problem)
  - one answer is to sort n-1, then just slot the last element into the sorted order (insertion sort)
  - another answer is for you to pick the smallest single entry, then give remaining elements to your friend to sort and add your element to the beginning of her results (selection sort)
  - what if your friend can only sort things of size n/2 or smaller? She can sort the two pieces... can we quickly make a sorted list from what's left? (merge sort!)
Recursion (Top Down) Merge Sort (2/7)

- **Partition** sequence into two sub-sequences of N/2 elements
- Recursively **partition** and **sort** each sub-array
- **Merge** the sorted sub-arrays
Recursion (Top Down) Merge Sort (3/7)

- **Partition** sequence into two sub-sequences of N/2 elements
- Recursively **partition** and **sort** each sub-array
- **Merge** the sorted sub-arrays

Figure: Merge sort divide phase
public class Sorts {
    public ArrayList<Integer> mergeSort(ArrayList<Integer> list) {
        if (list.size() == 1) {
            return list;
        }
        int middle = list.size() / 2;
        ArrayList<Integer> left =
            this.mergeSort(list.subList(0, middle));
        ArrayList<Integer> right =
            this.mergeSort(list.subList(middle, list.size()));
        return this.merge(left, right);
    }
    //code for merge() coming next!
}

ArrayList list is the sequence to sort, a sequence of ints
Base case: return the list when you get to its last element
Else recur on both halves of the list and merge the sorted lists
public class Sorts {
    public ArrayList<Integer> mergeSort(ArrayList<Integer> list) {
        if (list.size() == 1) {
            return list;
        }
        int middle = list.size() / 2;
        ArrayList<Integer> left =
            this.mergeSort(list.subList(0, middle));
        ArrayList<Integer> right =
            this.mergeSort(list.subList(middle, list.size()));
        return this.merge(left, right);
    }
    // code for merge() coming next!
}
public ArrayList merge(ArrayList<Integer> A, ArrayList<Integer> B) {
    ArrayList<Integer> result = new ArrayList<Integer>();
    int aIndex = 0;
    int bIndex = 0;
    while (aIndex < A.size() && bIndex < B.size()) {
        if (A.get(aIndex) <= B.get(bIndex)) {
            result.add(A.get(aIndex));
            aIndex++;
        } else {
            result.add(B.get(bIndex));
            bIndex++;
        }
    }
    if (aIndex < A.size()) {
        result.addAll(A.subList(aIndex, A.size()));
    }
    if (bIndex < B.size()) {
        result.addAll(B.subList(bIndex, B.size()));
    }
    return result;
}
Recursive (Top Down) Merge Sort (7/7)

- Each level of the tree performs \( n \) operations to merge and sort the subproblems below it.
- Each time you merge, you have to handle all the elements of the sub-arrays you’re merging, hence \( O(N) \). Recursion adds to runtime, but not much.
- There are \( \log_2 N \) merge passes.
- Thus, \( O(N \log_2 N) \) — way better than \( O(N^2) \).
  - can also drop log base (2) and say \( O(N \log N) \), since we can ignore constants.
- Learn much more about how to find the runtime of these types of algorithms in CS200!
Iterative (Bottom Up) Merge Sort

- Merge sort can also be implemented iteratively… non-recursive!

- Loop through array of size N, sorting 2 items each. Loop through the array again, combining the 2 sorted items into sorted item of size 4. Repeat, until there is a single item of size N!

- Number of iterations is $\log_2 N$, rounded up to nearest integer. 1000 elements in the list, only 10 iterations!

- Iterative merge sort avoids the nested method invocations caused by recursion!
Comparing Sorting Algorithms

Bubble Sort – $O(N^2)$

Insertion Sort – $O(N^2)$

Merge Sort - $(N\log_2 N)$
TopHat Question

Which sorting algorithm that we have looked at is the fastest (in terms of \textbf{worst-case} runtime)?

A. Bubble Sort
B. Insertion Sort
C. Merge Sort
D. Selection Sort
That’s It!

● Runtime is a very important part of algorithm analysis!
  o worst case runtime is what we generally focus on
  o know the difference between constant, linear, and quadratic run-time
  o calculate/define runtime in terms of Big-O Notation

● Sorting!
  o runtime analysis is very significant for sorting algorithms
  o types of simple sorting algorithms - bubble, insertion, selection, merge sort
  o fancier sorts perform even better, but tough to analyze, e.g., QuickSort
  o different algorithms have different performances and time complexities
What’s next?

● You have now seen how different approaches to solving problems can dramatically affect speed of algorithms
  ○ this lecture utilized arrays to solve most problems

● Subsequent lectures will introduce more data structures beyond arrays and arraylists that can be used to handle collections of data

● We can use our newfound knowledge of algorithm analysis to strategically choose different data structures to further speed up algorithms!
Announcements

- **DoodleJump late deadline tomorrow 11/5 @ 11:59pm**

- **DoodleJump Individual Check-Ins**
  - sign up on [this form](#) after submitting the project
  - you’ll receive an email the day of your check-in with the location – please show up 2-3 minutes early so we can stay on schedule
  - remember you must attend a check-in to reach MF on the assignment (i.e., to pass the course)

- **Tetris partners** are out
  - please check to make sure you’re listed correctly
  - Tetris will be out on Sunday – you do NOT want to procrastinate on this assignment!
  - the earlier you start, the shorter the lines at debugging hours 😊

- **Next week is the final lab/section of the semester**
Topics in Socially-Responsible Computing

Labor II
Labor in the tech industry

• Tech is largely not unionized because:
  • engineers/product managers/designers make a lot of money
  • people don’t work at companies for long enough (often 1-2 years) to have meaningful tie to them

• Huge surge in tech labor organizing in last few years

• Many of you will choose to work in industry as engineers/PMs/etc and your labor will be in high demand!

• Companies cannot build products if they can’t hire people OR if employees resist working on a project!

• We talk about labor because it is a way (that has succeeded!) to push for change within companies beyond just asking!
NYT Tech Workers Union

- Angie Kim ’20
  - former machine learning engineer at the New York Times
  - while there, organizing committee for New York Times Tech Guild
  - my technical interviewing workshop mentor!
- Goals (abbreviated)
  - pay transparency and minimum salaries for positions (often huge disparity between how people are paid for same job based on race/gender/etc)
  - better healthcare
  - more accountability on diversity and commitment to better recruitment
- Can bargain collectively on behalf of everyone within union — if union approves a strike, puts pressure on company to make a deal
- Process:
  - group of organizers reach out to people within the “bargaining unit” (in NYT Tech Guild case, 700 people)
  - collect “union cards” from people in group — once they pass 50% of workers, management can either voluntarily recognize union as legitimate or it goes to a National Labor Relations Board election (secret ballots, company often tries to undermine support)
  - NLRB elections take 4-6 months, currently in process

Credit: Reboot

Angie Kim. Credit: Sustainable Horizons Institute
Google Labor Actions (1/2)

• Project Maven (2018)
  • intended Pentagon project – would use AI to differentiate people/things in drone footage
  • thousands of employees signed internal letter, “we believe that Google should not be in the business of war,” invoked Google’s internal model of ‘don’t be evil’
  • after protest, Google announced they would not renew their contract with the Department of Defense

• Project Dragonfly (2018)
  • plan for Google to build a censored search engine for China that complies with strict Chinese govt censorship rules
  • “Dragonfly would also enable censorship and government-directed disinformation, and destabilize the ground truth on which popular deliberation and dissent rely” — protest letter signed by 1000 employees
  • Google terminated the project in 2019
Google Labor Actions (2/2)

• 2018 Google Walkout
  • Google paid Android creator a $90 million severance package after internally determining a sexual harassment claim against him was valid
  • 20,000 Google participated to protest handling of sexual assault more generally (The Verge)
  • Google agreed to several of their demands, including no longer forcing employees to settle disputes with the company in private arbitration

• Alphabet Workers Union (2021)
  • Alphabet, Google’s parent company
  • “minority union” — unlike NYT union, not a majority of workers, currently >800 members
  • “solidarity union” — not just full time employees, also part time employees, contractors, vendors, etc
Facebook Labor Actions

The New York Times

Facebook Employees Stage Virtual Walkout to Protest Trump Posts

While Twitter started labeling some of the president’s inflammatory messages, Facebook’s chief executive, Mark Zuckerberg, has said his company should leave them alone.


"'It just feels like, once again, we are erring on the side of a populist government and making decisions due to politics, not policies,' one worker wrote in an internal message that was reviewed by The Times.” — NYT, 2021

India and Israel Inflame Facebook’s Fights With Its Own Employees

The social network wrongly bowed to government demands to take down content in the countries, employees said, in more signs of internal dissent.

Other organizing

• Mijente: #NoTechForICE Campaign
  • Palantir, defense contractor, operates platform that aids in the location and deportation of undocumented people in the US
  • protests at Brown, Stanford, Carnegie Mellon, University of Washington, Harvard, etc. signed petition

• Palantir once part of Brown’s “Industry Partners Program”
  • program that “offers […] opportunities to collaborate with faculty, learn about Brown's research, and meet Brown students who are looking for employment.”
  • after career fair protest, partnership “paused,” now not a partner

• You have power to push your employer to do the right thing!
Recent News

• Oct 20, 2021: walkout at Netflix over airing of recent Dave Chapelle special, alleging transphobia

• Nov 2, 2021: Facebook shut down its facial recognition software (only on flagship FB) after collecting faceprints of more than 1 billion people

• Nov 3, 2021 (yesterday!): “Google Wants to Work with the Pentagon Again, Despite Employee Concerns” — New York Times
More reading that may be of interest!

- "Google Urged the U.S. to Limit Protection for Activist Workers" (2019) – Bloomberg
- "Let’s build the future of journalism together" — NYT Tech Guilt
- “One year after the Google walkout, key organizers reflect on the risk to their careers” (2020) – CNN
- "Google broke US law by firing workers behind protests, complaint says" (2020) — The Guardian
- Letter to Google CEO
- "Google Hedges on Promise to End Controversial Involvement in Military Drone Contract" (2019) — The Intercept
- "Incorporating Google’s AI Principles into Google Cloud" (2018) — Google
- "Google hired microworkers to train its controversial Project Maven AI" (2019) — The Verge
- "Google reportedly leaving Project Maven military AI program after 2019" (2018) — The Verge
- "Google employees demand the company pull out of Pentagon AI project" (2018) — The Verge
- Alphabet Workers Union
- "India and Israel Inflame Facebook’s Fights With Its Own Employees" (2021) — New York Times
- “Netflix employees stage a trans solidarity walkout, pose list of demands” (2021) — TechCrunch
- "Facebook is shutting down its facial recognition software" (2021) — CNN