CS260 – Lecture 6

Yan Gu

# Algorithm Engineering
(aka. How to Write Fast Code)

# I/O Algorithms and Parallel Samplesort

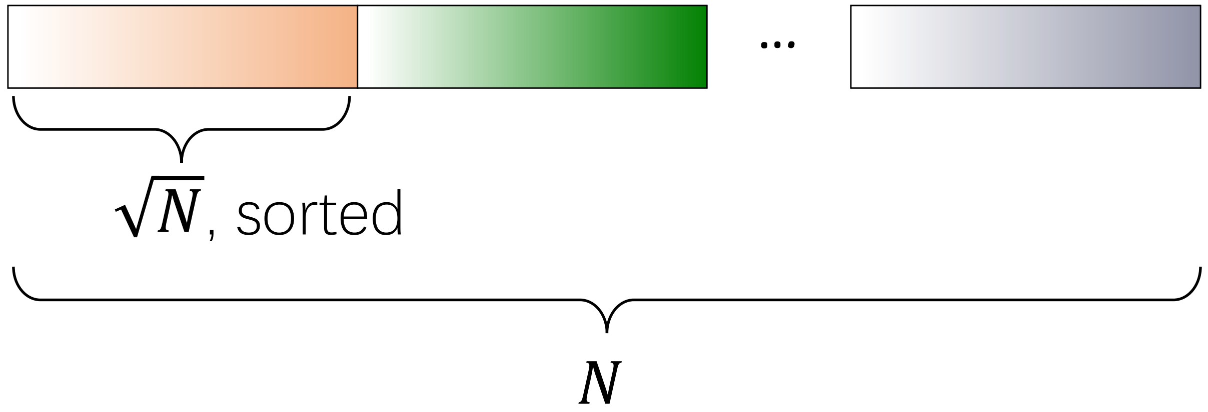
# CS260: Algorithm Engineering Lecture 6

- Review of Samplesort
- Semisort
- Course Policy

# Sample-sort outline

Analogous to multiway quicksort

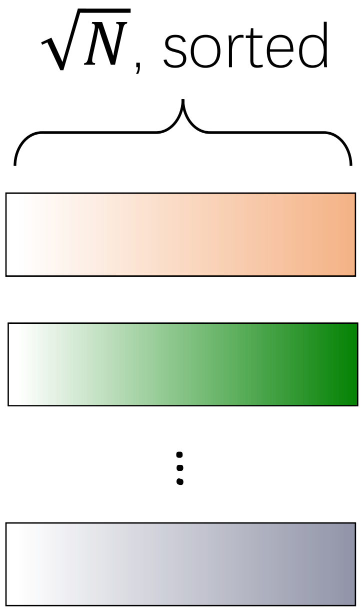
1. Split input array into $\sqrt{N}$ contiguous *subarrays* of size $\sqrt{N}$. Sort subarrays recursively



# Sample-sort outline

Analogous to multiway quicksort

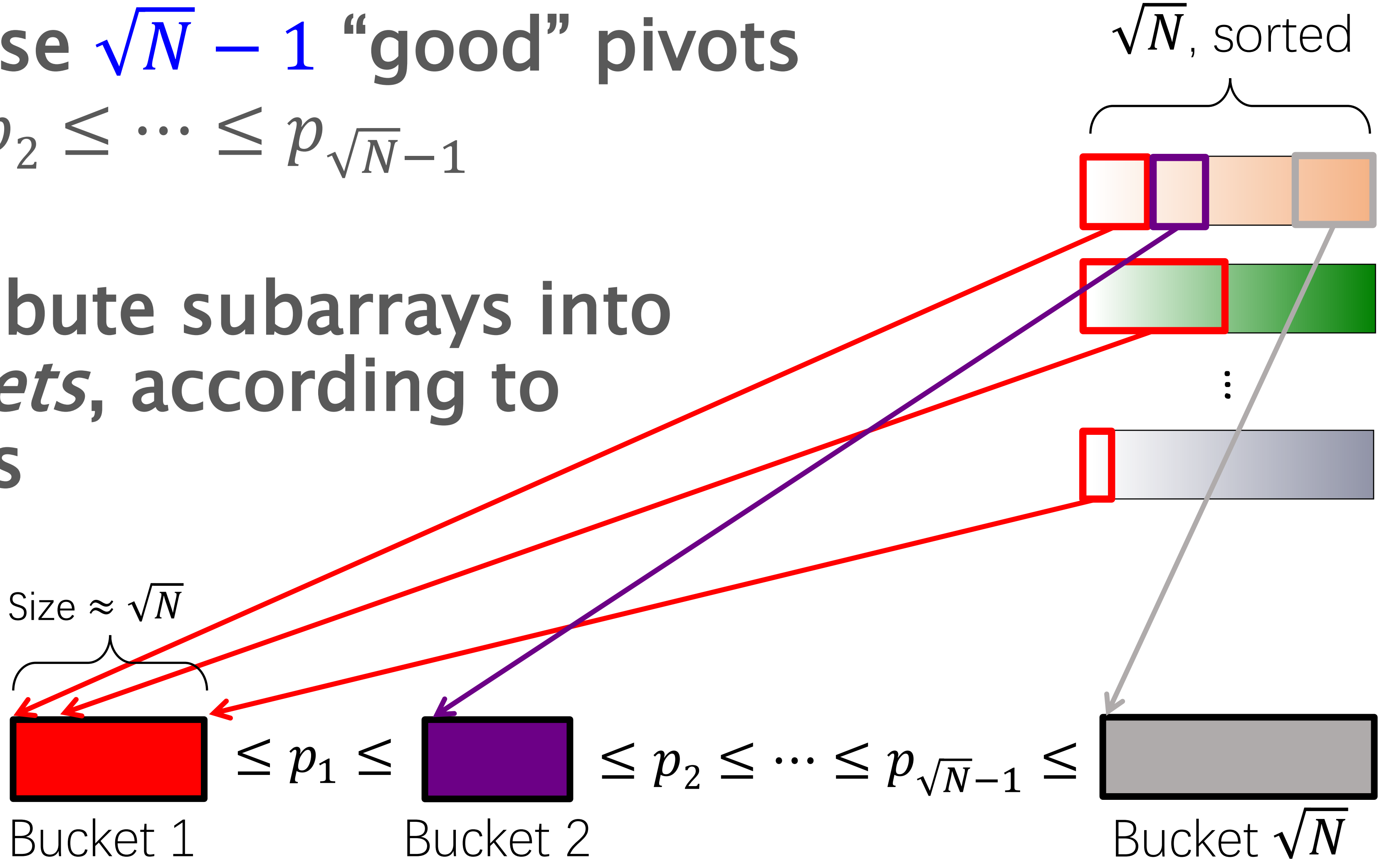
1. Split input array into $\sqrt{N}$ contiguous *subarrays* of size $\sqrt{N}$. Sort subarrays recursively (sequentially)



# Sample-sort outline

2. Choose $\sqrt{N} - 1$ "good" pivots
   $p_1 \leq p_2 \leq \cdots \leq p_{\sqrt{N}-1}$

3. Distribute subarrays into *buckets*, according to pivots



# Sample-sort outline

## 4. Recursively sort the buckets

Bucket 1 $\leq p_1 \leq$ Bucket 2 $\leq p_2 \leq \cdots \leq p_{\sqrt{N}-1} \leq$ Bucket $\sqrt{N}$

## 5. Copy concatenated buckets back to input array

sorted

# CS260: Algorithm Engineering Lecture 6

Review of Samplesort

Semisort

Course Policy

# What is semisort?

| key | 45 | 12 | 45 | 61 | 28 | 61 | 61 | 45 | 28 | 45 |
|---|---|---|---|---|---|---|---|---|---|---|
| Value | 2 | 5 | 3 | 9 | 5 | 9 | 8 | 1 | 7 | 5 |

- **Input:**
  - An array of records with associated keys
  - Assume keys can be hashed to the range $[n^k]$
- **Goal:**
  - All records with equal keys should be adjacent

# What is semisort?

| key | 12 | 61 | 61 | 61 | 45 | 45 | 45 | 45 | 28 | 28 |
|---|---|---|---|---|---|---|---|---|---|---|
| Value | 5 | 8 | 9 | 9 | 2 | 5 | 1 | 3 | 7 | 5 |

- **Input:**
  - An array of records with associated keys
  - Assume keys can be hashed to the range $[n^k]$
- **Goal:**
  - All records with equal keys should be adjacent

# What is semisort?

| key | 45 | 45 | 45 | 45 | 12 | 61 | 61 | 61 | 28 | 28 |
|---|---|---|---|---|---|---|---|---|---|---|
| Value | 2 | 5 | 1 | 3 | 5 | 8 | 9 | 9 | 7 | 5 |

- **Input:**
  - An array of records with associated keys
  - Assume keys can be hashed to the range $[n^k]$
- **Goal:**
  - All records with equal keys should be adjacent
  - Different keys are not necessarily sorted
  - Records with equal keys do not need to be sorted by their values

# What is semisort?

| key | 45 | 45 | 45 | 45 | 12 | 61 | 61 | 61 | 28 | 28 |
|---|---|---|---|---|---|---|---|---|---|---|
| Value | 1 | 5 | 3 | 2 | 5 | 8 | 9 | 9 | 7 | 5 |

- **Input:**
  - An array of records with associated keys
  - Assume keys can be hashed to the range $[n^k]$
- **Goal:**
  - All records with equal keys should be adjacent
  - Different keys are not necessarily sorted
  - Records with equal keys do not need to be sorted by their values

# Semisort is one of the most useful primitives in parallel algorithms

| |
|---|
| Parallel In-Place Algorithms: Theory and Practice |
| Julienne: A Framework for Parallel Graph Algorithms using Work-efficient Bucketing |
| Semi-Asymmetric Parallel Graph Algorithms for NVRAMs |
| Efficient BVH Construction via Approximate Agglomerative Clustering |
| Theoretically-Efficient and Practical Parallel DBSCAN |

# Why is semisort so useful? (albeit not seen before)

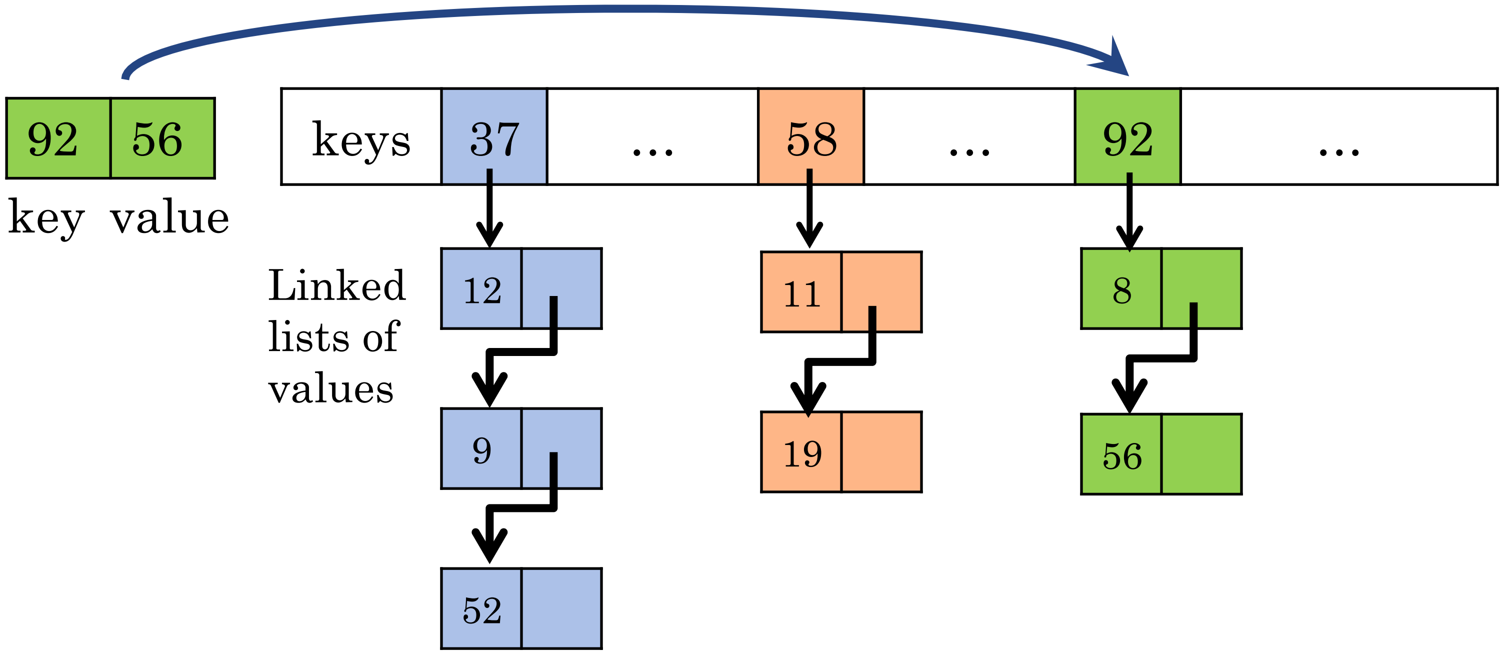
- **Semisorting can be done by sorting, but faster (less restriction)**
  - Theoretically can be done in $O(n)$ work not $O(n \log n)$ work

- **Can be used to implement counting / integer sort**
  - Integer sort: given $n$ key-value pairs with keys in range $[1, \ldots, n]$, query the KV-pairs with a certain key
  - Counting sort: given $n$ key-value pairs with keys in range $[1, \ldots, n]$, query the number of KV-pairs with a certain key
  - In database community, this is called the `GroupBy` operator

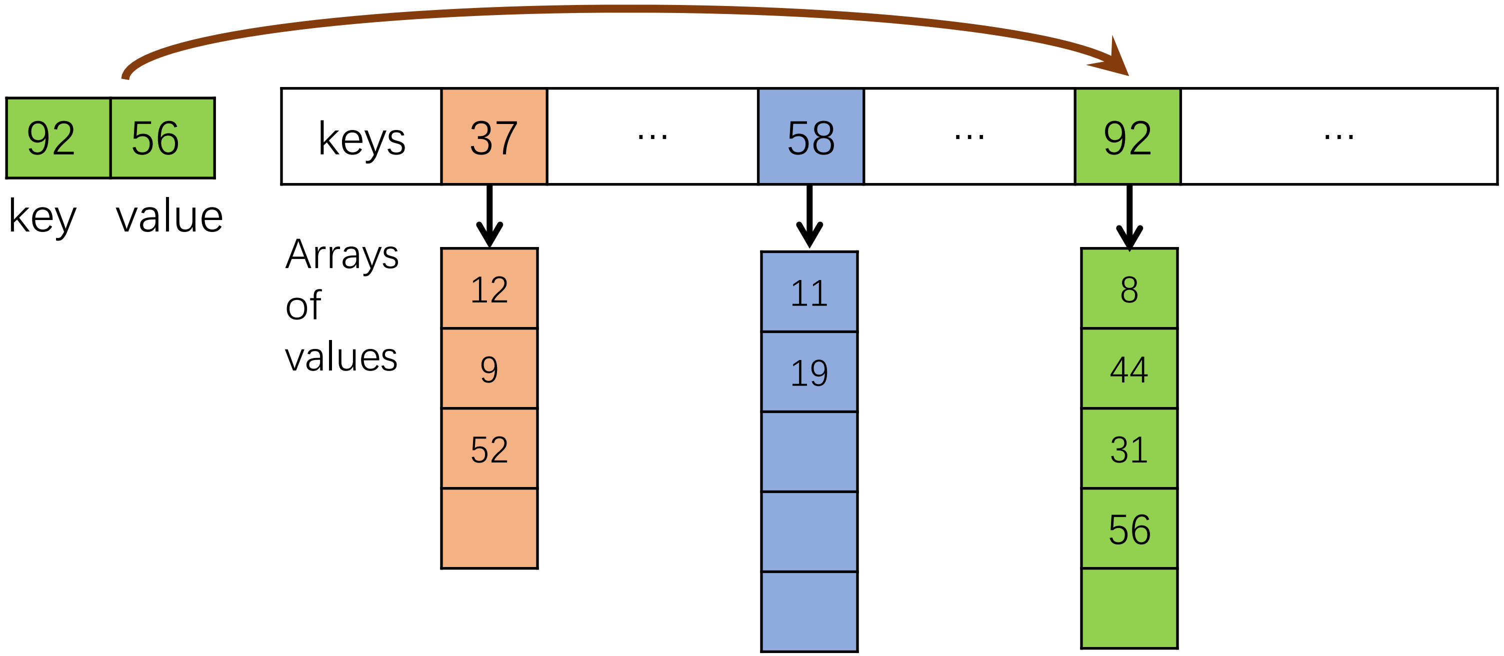
# Why is semisort so useful? (albeit not seen before)

- **Semisorting can be done by sorting, but faster (less restriction)**
  - Theoretically can be done in $O(n)$ work not $O(n \log n)$ work

- **Can be used to implement counting / integer sort**



# Attempts – Sequentially: Pre-allocated array



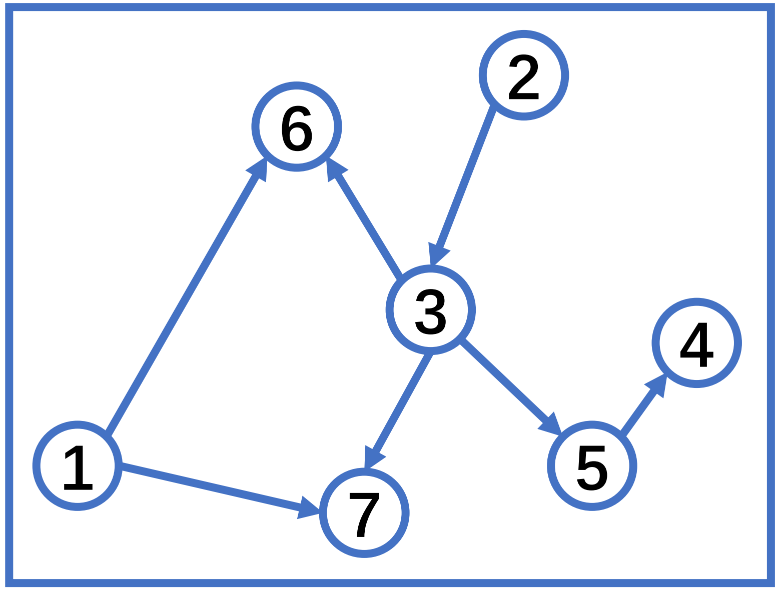

- Problem
  - Need to pre-count the number of each key

# Another use case for semisrot

- **Generate adjacency array for a graph**

| Edge list |
|---|
| (3,5) |
| (1,7) |
| (2,3) |
| (3,6) |
| (5,4) |
| (3,7) |
| (1,6) |

| Sorted edge list |
|---|
| (3,5) |
| (3,7) |
| (3,6) |
| (5,4) |
| (1,6) |
| (1,7) |
| (2,3) |



# What is semisort?

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| key | 45 | 45 | 45 | 45 | 12 | 61 | 61 | 61 | 28 | 28 |
| Value | 1 | 5 | 3 | 2 | 5 | 8 | 9 | 9 | 7 | 5 |

- **Input:**
  - An array of records with associated keys
  - Assume keys can be hashed to the range $[n^k]$
- **Goal:**
  - All records with equal keys should be adjacent
  - Different keys are not necessarily sorted
  - Records with equal keys do not need to be sorted by their values

# Why is semisort hard?

| key | 45 | 45 | 45 | 45 | 12 | 61 | 61 | 61 | 28 | 28 |
|---|---|---|---|---|---|---|---|---|---|---|
| Value | 1 | 5 | 3 | 2 | 5 | 8 | 9 | 9 | 7 | 5 |

- **There can be many duplicate keys**
  - Heavy keys
- **Or, there can be almost no duplicate keys**
  - Light keys

# Implement integer sort using semisort

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| key | 45 | 45 | 45 | 45 | 12 | 61 | 61 | 61 | 28 | 28 |
| Value | 1 | 5 | 3 | 2 | 5 | 8 | 9 | 9 | 7 | 5 |

- **Input: $n$ KV-pairs with key in $[n]$**
- **Step 1: hash the keys (i.e., for $(k_i, v_i)$, generate $h_i = \text{hash}(k_i)$)**
- **Step 2: semisort $(h_i, (k_i, v_i))$, and resolve conflicts**
- **Step 3: get the pointer for each key $k_i$**

# The Top-Down Parallel Semisort Algorithm

# The main goal estimate key counts

- **And tell the heavy keys from light ones.  By how?**

**Sampling!**

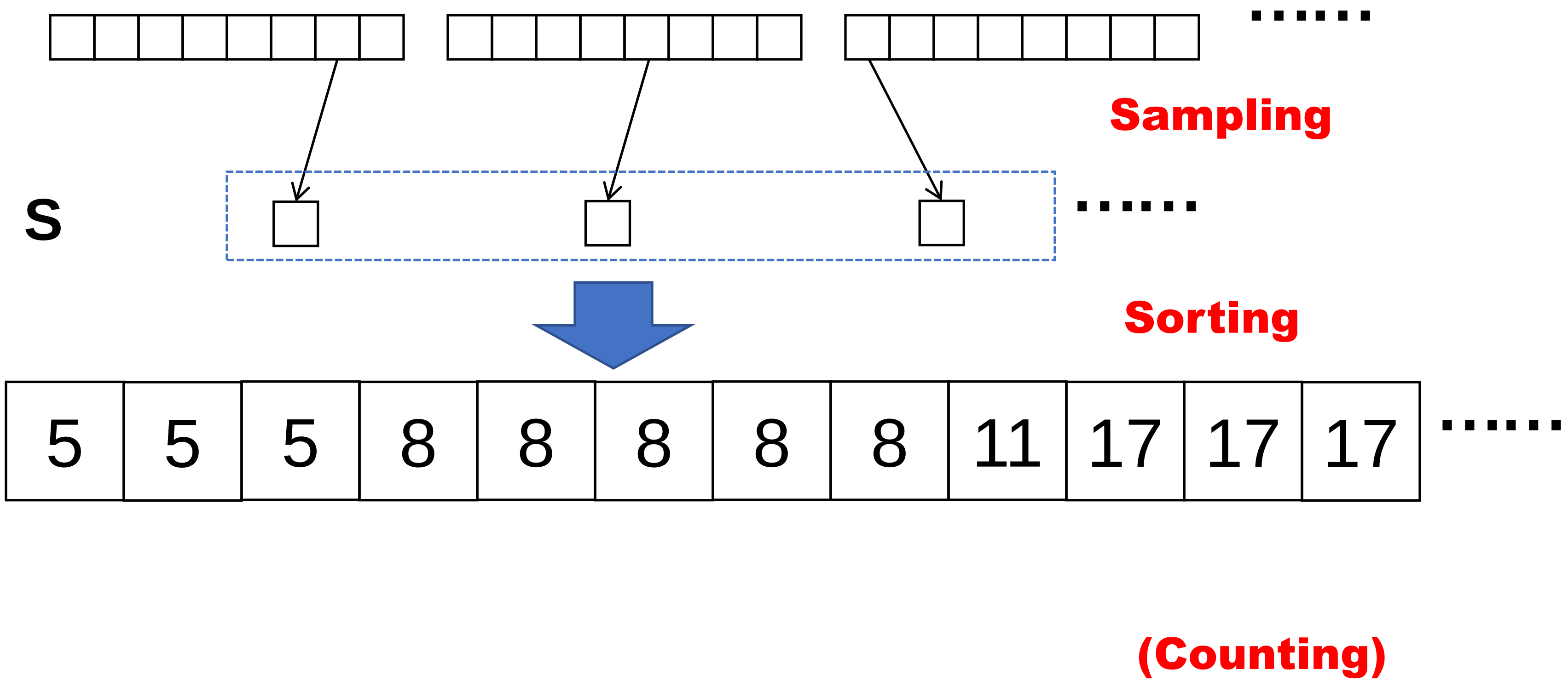
- **For a key appear more than $n/t$ times, we call it a heavy key**
- **Otherwise, we call it a light key**
- **We can treat them separately**

# The algorithm

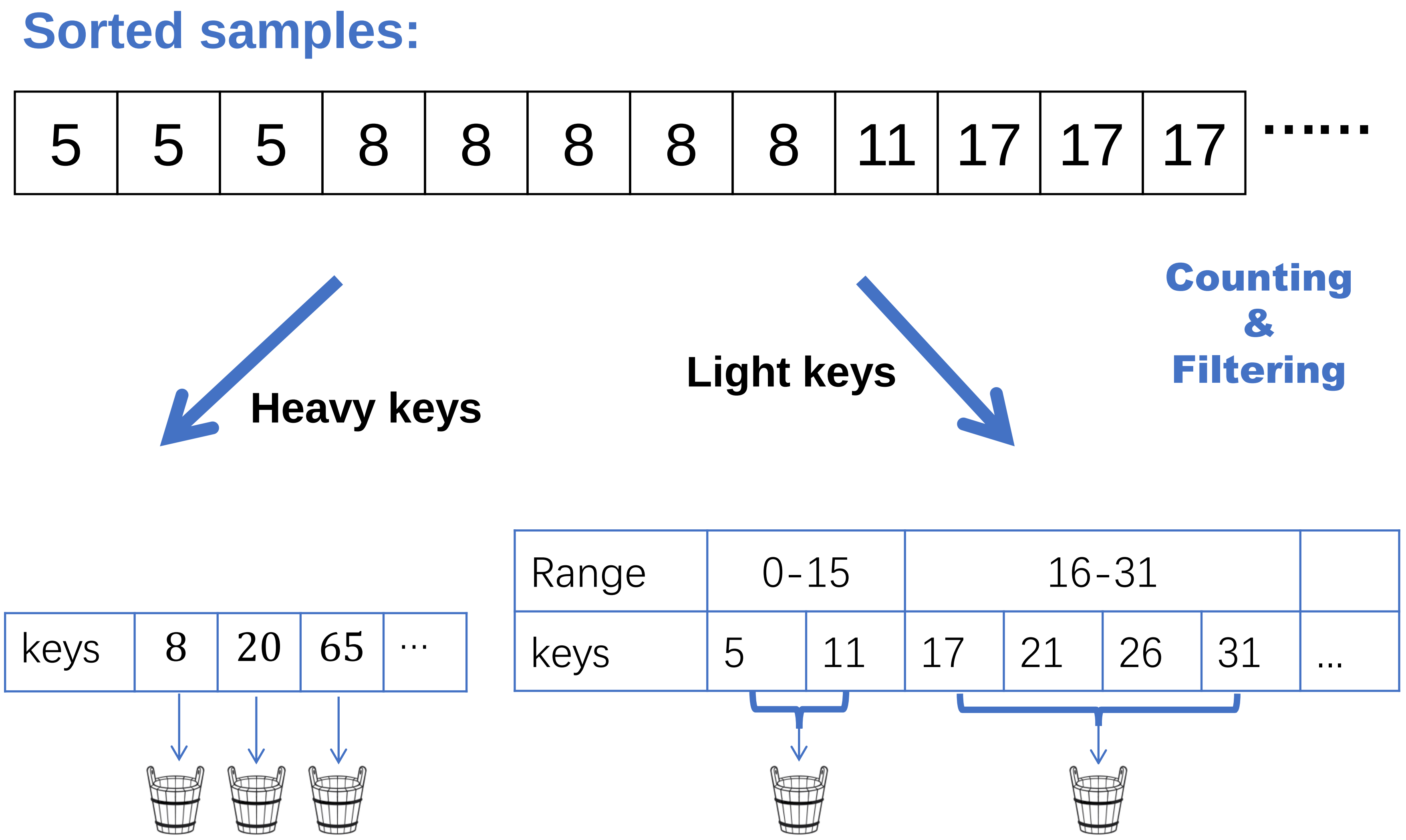
- **Take $t \log n$ samples and sort them**
- **For those keys with more than $\log n$ appearances, we mark them as heavy keys, others are light keys**
- **We give each heavy key a bucket, and the another $t$ buckets for light keys each corresponds to a range of $n^k/t$**
  - The input keys are hashed into $[n^k]$
  - In total we have no more than $2t$ buckets
  - The rest of the algorithm is pretty similar to samplesort

# Phase 1: Sampling and sorting

1. Select a sample set $S$ with $t \log n$ of keys
2. Sort $S$



# Phase 2: Bucket Construction



# At the end of Phase 2

- **In total we have no more than $2t$ buckets**
  - $t$ of them are for light keys

- **Then we construct a hash table for the heavy keys**

- **Now we know which bucket each KV-pair $(k_i, v_i)$ goes to:**
  - If $k_i$ is found in the hash table, assign it to the associated heavy bucket
  - Otherwise, it goes to the light bucket based on the range of $k_i$

- **The rest of the algorithm is almost identical to samplesort**

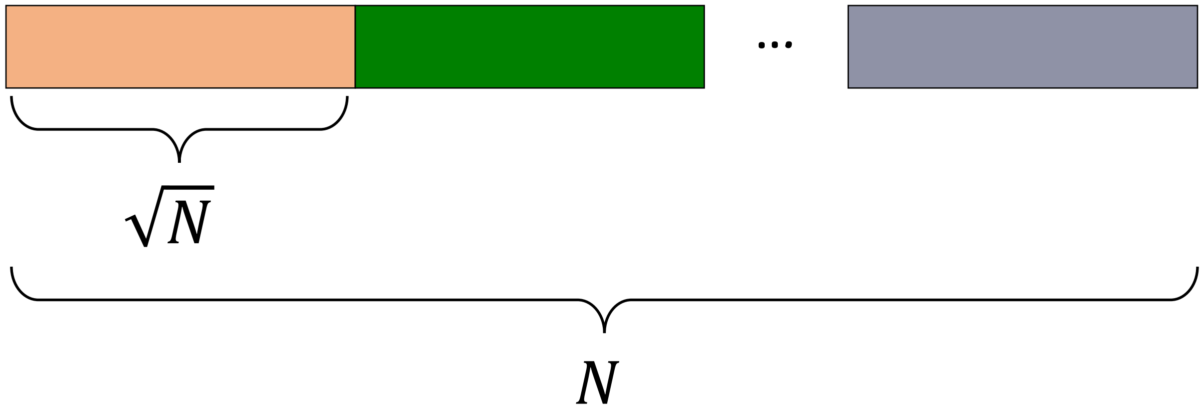
# Sample-sort outline

**Analogous to multiway quicksort**

1. **Split input array into $\sqrt{N}$ contiguous *subarrays* of size $\sqrt{N}$**

$N/t$

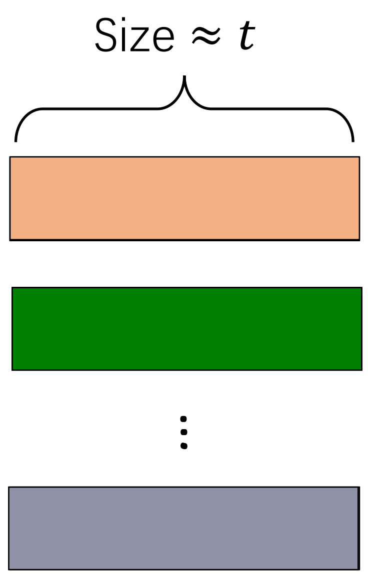
$t$



# Sample-sort outline

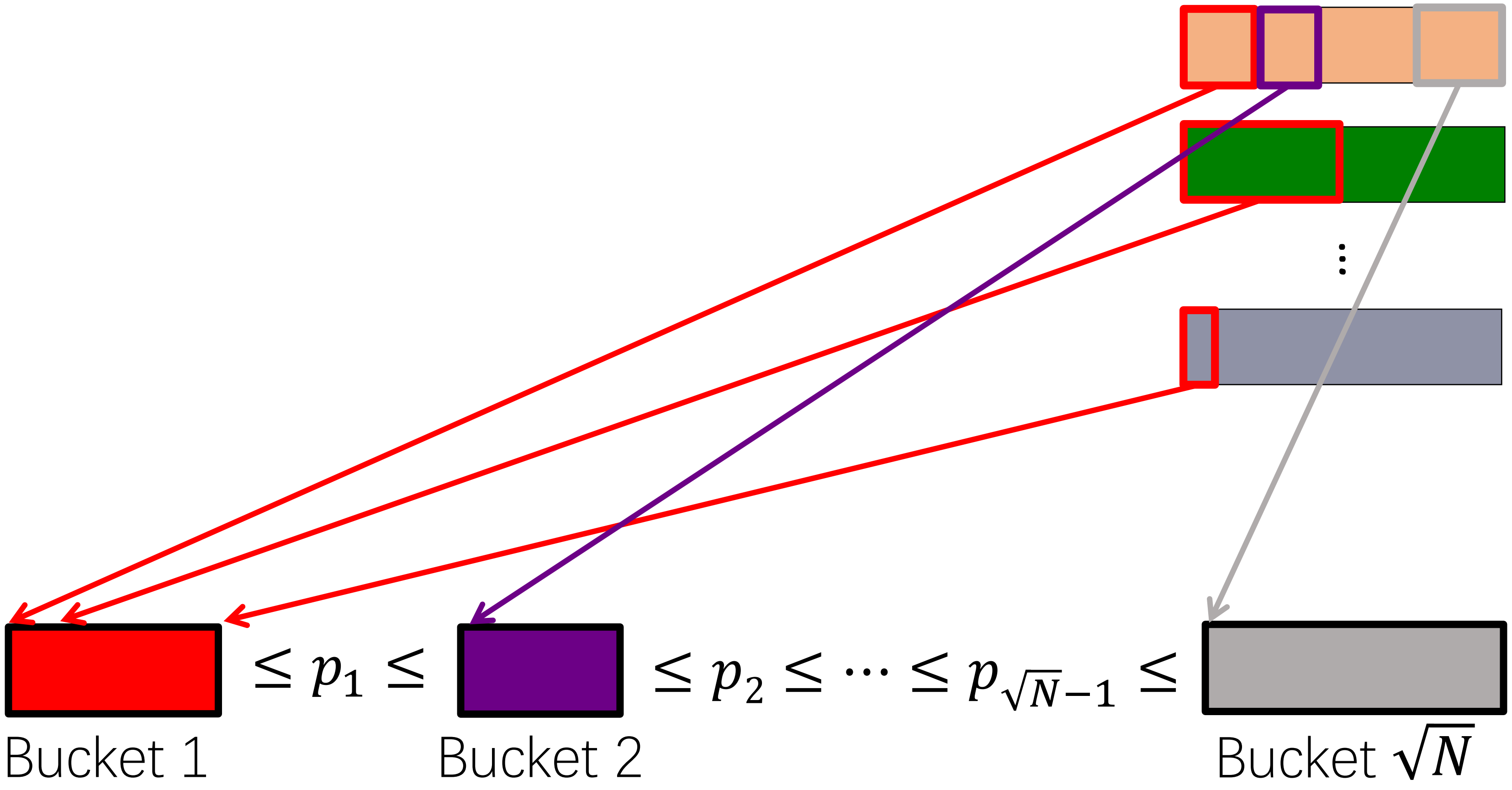
**Analogous to multiway quicksort**

1. **Split input array into $N/t$ contiguous *subarrays* of size $t$. Sort subarrays recursively (sequentially)**



# Sample-sort outline

2. Distribute subarrays into *buckets*



# Sample-sort outline

**Only for the light buckets**

## 3. Recursively sort the buckets


## 4. Copy concatenated buckets back to input array

# Difference 2: subarrays are not sorted

- For simplicity, assume $n = 16$, and the input is
$$[1,2,3,4, \quad 1,1,3,3, \quad 1,2,2,4, \quad 1,2,4,4]$$
- First, get the count for each subarray in each bucket
$$[1,1,1,1, \quad 2,0,2,0, \quad 1,2,0,1, \quad 1,1,0,2]$$
- Then, transpose the array and scan to compute the offsets
$$[1,2,1,1, \quad 1,0,2,1, \quad 1,2,0,0, \quad 1,0,1,2]$$
$$[0,1,3,4, \quad 5,6,6,8, \quad 9,10,12,12, \quad 12,13,13,14]$$
- Lastly, move each element to the corresponding bucket
$$[1,\mathbf{1},\mathbf{1},\emptyset,\emptyset,2,\emptyset,\emptyset,\emptyset,3,\mathbf{3},\mathbf{3},4,\emptyset,\emptyset,\emptyset]$$

# Difference 2: subarrays are not sorted

- **For simplicity, assume $n = 16$, and the input is**

$$[1, \mathbf{3}, \mathbf{2}, 4, \quad 1, \mathbf{3}, \mathbf{1}, 3, \quad 1, 2, 2, 4, \quad 1, 2, 4, 4]$$

- **First, get the count for each subarray in each bucket**

$$[1, 1, 1, 1, \quad 2, 0, 2, 0, \quad 1, 2, 0, 1, \quad 1, 1, 0, 2]$$

- **Then, transpose the array and scan to compute the offsets**

$$[1, 2, 1, 1, \quad 1, 0, 2, 1, \quad 1, 2, 0, 0, \quad 1, 0, 1, 2]$$

$$[0, 1, 3, 4, \quad 5, 6, 6, 8, \quad 9, 10, 12, 12, \quad 12, 13, 13, 14]$$

- **Lastly, move each element to the corresponding bucket**

$$[1, \mathbf{1}, \mathbf{1}, \emptyset, \emptyset, 2, \emptyset, \emptyset, \emptyset, 3, \mathbf{3}, \mathbf{3}, 4, \emptyset, \emptyset, \emptyset]$$

# Take away for semisort

- **Semisort is very useful**
  - Implements bucket and integer sort, and can apply on even large key range
  - Theoretically takes linear work and $O(\log n)$ depth, although in this lecture I talked about a simpler version that does not have either bound
- **The key insight is the partition of heavy and light keys**
  - Heavy keys have own buckets, which can be large but need no further sort
  - Light keys are grouped based on ranges. Since the keys are hashed, the light buckets are small (contains $O(n/t)$ elements, analysis in **[GSSB15]**)

# CS260: Algorithm Engineering Lecture 6

Review of Samplesort

Semisort

Course Policy

# Paper Reading and Course Presentation

# Paper Reading and Course Presentation

- 10 students have reserved the papers for reading and presenting
- If Paper 8 is not reserved, Yunshu will present it on 4/27
- Deadlines, instructions and schedules are on course webpage and ilearn

# Course Presentation

- Each of you will give a 22-minute talk and have a 5-minute Q&A.  Time management is crucial.

- Tomorrow, I will upload Prof. Sun's lecture on ***how to give a clear talk***.  It is mandatory to study it before your presentation.

- Meanwhile, I will also attach a *speaking skill evaluation* form that is used to evaluate your talk.
    - You should check it before you give the presentation

# Preparation for Course Presentation

- It's highly recommended to give 2-3 practice talks to your friends and/or classmates before your presentation, in order to guarantee that everything you say makes sense and is understandable.

- Otherwise you are just wasting everyone's time.  Let's don't do it since it's embarrassing.

- You're obliged to submit mostly-done slides to Yan 48h ahead, as well as the corresponding paper reading.

# Quiz

# Quiz

- Quiz is on 4/24.  I will send each of you a google doc, and you should answer in it.
- Don't write in other apps and copy and paste to that.  Google doc keeps track of all your edits (so please don't cheat).
  - Cheating the quiz/exam is fatal.  Please don't let me handle that.
- Only for 10% score.  Don't panic.
- It is open-book, but you should still review the lectures since the length is for 1 hour and you might not have time to search for each problem.

# Midterm and Final Project

# Midterm Project

- **Due on April 29, so you still have more than 2 weeks.**
- **It's a hard deadline, if you feel short of time, submit what you have at that time**
  - Pre-proposal meeting is on May 1, and final proposal is due on May 4
- **You should start now, and meanwhile, You should expect at least two days in writing the report**
  - Writing a good report can largely increase your score

# Final Project

- Pre-proposal meeting: 5/1
- Proposal: 5/4
- Weekly progress report 1: 5/13
- Milestone: 5/22
- Weekly progress report 2: 5/29
- Final project presentation: 6/1-5
- Final report due: 6/8

# Final Project: Score Breakdown

- Proposal: 10%
- Weekly progress report 1: 5%
- Milestone: 10%
- Weekly progress report 2: 5%
- Final project presentation: 20%
- Final report: 50%

# Milestone and Final project

- Milestone: 5-minute talk for each student, discuss the progress and if you meet the goals in the proposal

- Final project presentation: 20+5(Q&A) minutes for each student, talk about your work like the paper presentation