CECS 274: Data Structures Sorting

Oscar Morales Ponce

California State University Long Beach

Problem motivation

- Suppose you work at Instagram and you need to have a program that answers queries such as:
  - What picture has the highest rate?
  - What picture has the second highest rate?
  - What picture has the n-highest rate?

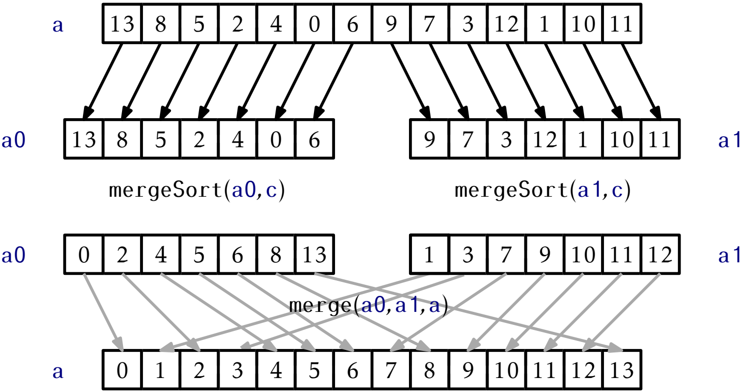
Comparison-Based Sorting

- Comparison-Based Sorting
  - Input
    - a: array storing the data of interest
    - n: number of elements
  - Output
    - a: such that $a[i] < a[i+1]$ for all $i < n-1$

Merge-Sort (Divide-and-Conquer)

- If the length of a is at most 1, then a is already sorted
- **Divide**: Otherwise, split a into two halves, $a_0 = a[0], \ldots, a[n/2-1]$ and $a_1 = a[n/2], \ldots, a[n-1]$ and recursively sort a_0 and a_1.
- **Conquer**: then merge (the now sorted) a_0 and a_1

Merge-Sort (Divide-and-Conquer)



Merge-Sort (Discussion Activity: Steps)

merge_sort(a)
- If the length of a is one, then it is sorted
- Let $m = ndiv2$
- Let $a_0 = a[0, ..., m-1]$ and $a_1 = a[m, ..., n-1]$
- Recursively sort a_0 and a_1
- Merge a_0 and a_1

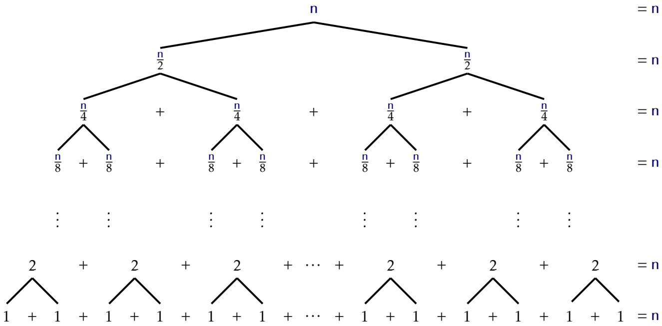
Merge-Sort: Merge

merge(a_0, a_1, a)
Let $i_0 = 0$ and $a_1 = 0$ be the current index in a_0 and a_1
For each i in the length of a
If we have visited all a_0 then $a[i] = a_1[i_1]$
if we have visited all a_1 then $a[i] = a_0[i_0]$
Otherwise let $a[i]$ be the smallest element of $a_0[i_0]$ and $a_1[i_1]$
Increase the index accordingly

Merge-Sort: Merge

merge(a_0, a_1, a)
    $i_0 \leftarrow i_1 \leftarrow 0$
    for i **in** $0, 1, 2, \ldots, \text{length}(a) - 1$ **do**
        if $i_0 = \text{length}(a_0)$ **then**
            $a[i] \leftarrow a_1[i_1]$
            $i_1 \leftarrow i_1 + 1$
        else if $i_1 = \text{length}(a_1)$
            $a[i] \leftarrow a_0[i_0]$
            $i_0 \leftarrow i_0 + 1$
        else if $a_0[i_0] \leq a_1[i_1]$
            $a[i] \leftarrow a_0[i_0]$
            $i_0 \leftarrow i_0 + 1$
        else
            $a[i] \leftarrow a_1[i_1]$
            $i_1 \leftarrow i_1 + 1$

Merge-Sort: Analysis



Merge-Sort: Complexity

Theorem

The $merge\_sort(a)$ *algorithm runs in* $O(n \log n)$ *time and performs at most* $O(n \log n)$ *comparisons.*

Proof.

- The proof is by induction on n
- The base case: $n = 1$ then it is trivial sorted.
- Inductive Step: Suppose it is true for $n/2$. We consider the case when n is even and n is odd independently
  - Let $C(n)$ denote the maximum number of comparisons performed by $merge\_sort(a)$ on an array a of length n

□

Merge-Sort: n is even

Merging two sorted lists of total length n requires at most $n-1$ comparations

$$
\begin{aligned}
C(n) &\leq n - 1 + 2C(n/2) \\
&\leq n - 1 + 2((n/2)\log(n/2)) \\
&= n - 1 + n\log(n/2) \\
&= n - 1 + n\log n - n \\
&< n\log n
\end{aligned}
$$

Merge-Sort: n is odd

Observe that for all $x \geq 1$

$$\log(x+1) \leq \log(x) + 1$$

and for all $x \geq 1/2$

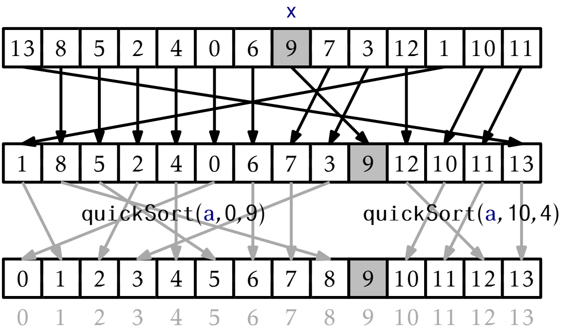
$$\log(x+1/2) + \log(x-1/2) \leq 2\log(x)$$

$$\begin{aligned}
C(n) &\leq n - 1 + C(\lceil n/2 \rceil) + C(\lfloor n/2 \rfloor) \\
&\leq n - 1 + \lceil n/2 \rceil \log\lceil n/2 \rceil + \lfloor n/2 \rfloor \log\lfloor n/2 \rfloor \\
&= n - 1 + (n/2 + 1/2)\log(n/2 + 1/2) + (n/2 - 1/2)\log(n/2 - 1/2) \\
&\leq n - 1 + n\log(n/2) + (1/2)(\log(n/2 + 1/2) - \log(n/2 - 1/2)) \\
&\leq n - 1 + n\log(n/2) + 1/2 \\
&< n + n\log(n/2) \\
&= n + n(\log n - 1) \\
&= n\log n
\end{aligned}$$

Quick-sort (Divide-and-Conquer)

- Pick a random *pivot* element, x, from a
- **Divide**: Partition a into the set of elements less than x, the set of elements equal to x, and the set of elements greater than x
- **Conquer**: Recursively sort the first and third sets in this partition

Quick-sort (Divide-and-Conquer)



Quick-Sort (Divide-and-Conquer)

- Invariant:

$$a[i] \begin{cases} < x & \text{if } 0 \leq i \leq p \\ = x & \text{if } p < i < q \\ > x & \text{if } q \leq i \leq n-1 \end{cases}$$

Quick-Sort (Divide-and-Conquer)

$\text{quick\_sort}(a, i, n)$
- If n is at most one, then it is sorted
- Let x be the value of a random index of a
- Let j be the current value of a. Initially i
- Initially $p = i - 1$ and $q = i + n$
- While $j < q$
  - If $a[j] < x$ then move to the first and update p and j
  - If $a[j] > x$ then move to the last and update q
  - If $a[j] = x$ then update j
- $\text{quick\_sort}(a, i, p - i + 1)$
- $\text{quick\_sort}(a, q, n - (q - i))$

Quick-Sort (Divide-and-Conquer)

```
quick_sort(a, i, n)
    if n ≤ 1 then return
    x ← a[i + random_int(n)]
    (p, j, q) ← (i − 1, i, i + n)
    while j < q do
        if a[j] < x then
            p ← p + 1
            a[j], a[p] ← a[p], a[j]
            j ← j + 1
        else if a[j] > x
            q ← q − 1
            a[j], a[q] ← a[q], a[j]
        else
            j ← j + 1
    quick_sort(a, i, p − i + 1)
    quick_sort(a, q, n − (q − i))

quick_sort(a)
    quick_sort(a, 0, length(a))
```

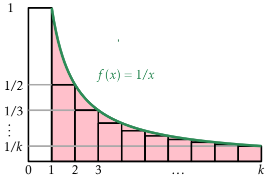
Quick-Sort: Harmonic number

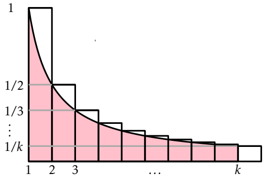
For a non-negative integer, k, the k-th *harmonic number*, denoted H_k, is defined as

$$H_k = 1 + 1/2 + 1/3 + \cdots + 1/k$$

The harmonic number H_k has no simple closed form, but it is very closely related to the natural logarithm of k. In particular,

$$\ln k < H_k \leq \ln k + 1$$





Quick-Sort: Analysis

Lemma

For any $x \in \{0, 1, , n-1\}$ the number of elements less than or equal x is H_{x+1} and the number of elements greater than or equal x is H_{n-x}.

Proof.

- Let $I_i = \begin{cases} 1 & a[i] \leq x \\ 0 & a[i] > x \end{cases}$
  Observe that $Pr[i \leq x] = \frac{1}{x-i+1}$
  $E[\sum_{i=0}^{n} I_i] = \sum_{i=0}^{n} E[I_i] = \sum_{i=0}^{n} \frac{1}{x-i+1} = H_{x+1} - 1$
- Let $I_i = \begin{cases} 1 & a[i] \geq x \\ 0 & a[i] < x \end{cases}$
  Observe that $Pr[i \geq x] = \frac{1}{i-x+1}$
  $E[\sum_{i=0}^{n} I_i] = \sum_{i=0}^{n} E[I_i] = \sum_{i=0}^{n} \frac{1}{i-x+1} = H_{n-x} - 1$

Quick-Sort: Analysis

Theorem

When quicksort is called to sort an array containing n distinct elements, the expected number of comparisons performed is at most $2n \ln n + O(n)$.

Proof.

Let T be the number of comparisons performed by quicksort when sorting n expectation, we have:

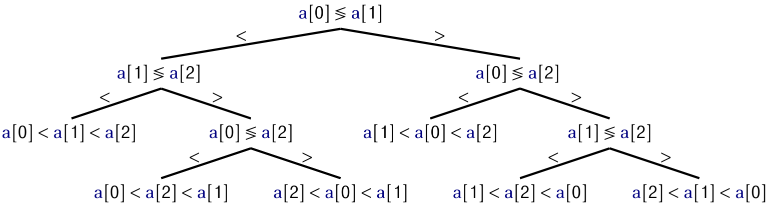
$$\begin{aligned}
\mathrm{E}[T] &= \sum_{i=0}^{n-1} (H_{i+1} + H_{n-i}) \\
&= 2\sum_{i=1}^{n} H_i \leq 2\sum_{i=1}^{n} H_n \\
&\leq 2n \ln n + 2n = 2n \ln n + O(n)
\end{aligned}$$

$\square$

Lower Bound

- If the only operations allowed on the elements of a are comparisons, then no algorithm can avoid doing roughly $n \log n$ comparisons.

$$\log(n!) = \log n + \log(n-1) + \cdots + \log(1) = n \log n - O(n)$$



Theorem

For any deterministic comparison-based sorting algorithm $\mathcal{A}$ and any integer $n \geq 1$, there exists an input array a of length n such that $\mathcal{A}$ performs at least $\log(n!) = n \log n - O(n)$ comparisons when sorting a.

Proof.

- The comparison tree defined by $\mathcal{A}$ must have at least $n!$ leaves.
- $\mathcal{A}$ has a leaf, w, with a depth of at least $\log(n!)$ and there is an input array a that leads to this leaf.

$\square$

Binary Search

- Binary Search
  - Input
    - x: key
    - n: number of elements
    - a: sorted array ($a[i] < a[i+1]$ for all $0 \leq i < n-1$)
  - Output
    - i: such that $a[i] = x$
  - Loop Invariant: if $x \in a[0, ..., n-1]$, then x is in $a[l, ..., r]$

Binary Search

$binary\_search(a, x)$
  Given an interval l and r, Initially, 0 and $n-1$, respectively
  While $r > l$
    Let $m = \lceil \frac{l+r}{2} \rceil$
    If $x \leq a[m]$ then search in $a[l, ..m]$
    Otherwise search in $a[m+1, n-1]$
  If x is equal $a[l]$ then return m
  Otherwise, x is not in a

Binary Search

$binary\_search(a, x)$
    $l = 0$
    $r = n - 1$
    while $r > l$
        $m = \lfloor \frac{l+r}{2} \rfloor$
        if $x \leq a[m]$ **then**
            $r = m$
        else
            $l = m + 1$
    if $a[l] = x$ **then**
        return l
    else
        "Key is not in a"

Binary Search: Analysis

Theorem

$binary\_search(a, x)$ runs in $O(\log n)$ time in the worst case

Proof.

Let t be the number of steps.

- In step 1, the interval is of size $n = \frac{n}{2^0}$
- In step 2, the interval is of size $\lfloor \frac{n}{2} \rfloor = \lfloor \frac{n}{2^1} \rfloor$
- In step 3, the interval is of size $\lfloor \frac{n}{4} \rfloor = \lfloor \frac{n}{2^2} \rfloor$

  $\vdots$

- In step t, the interval is of size $\lfloor \frac{n}{2^{t-1}} \leq 1 \rfloor$

Therefore, $n < 2^{(t+1)}$. Taking logarithms in both sides, $\log n \leq t + 1$

$\square$