Mergesort and Quicksort


Mergesort

- Divide array into two halves.
- Recursively sort each half.
- Merge two halves to make sorted whole.

input
M E R G E S O R T E X A M P L E
sort left
E E G M O R R S T E X A M P L E
sort right
E E G M O R R S A E E L M P T X
merge
A E E E E E G L M M O P R R S T X

Quicksort

- Java sort for primitive types.
- C qsort, Unix, g++, Visual C++, Perl, Python.

Mergesort and Quicksort

Two great sorting algorithms.

- Full scientific understanding of their properties has enabled us to hammer them into practical system sorts.
- Occupies a prominent place in world’s computational infrastructure.
- Quicksort honored as one of top 10 algorithms for science and engineering of 20th century.

Mergesort.

- Java sort for objects.
- Perl stable, Python stable.

Quicksort.

- Java sort for primitive types.
- C qsort, Unix, g++, Visual C++, Perl, Python.
Mergesort: Example

Mergesort: Java Implementation

```java
public class Merge {
    private static void sort(Comparable[] a, Comparable[] aux, int l, int m, int r) {
        if (r <= l) return;
        int m = (r + l) / 2;
        sort(a, aux, l, m);
        sort(a, aux, m + 1, r);
        merge(a, aux, l, m, r);
    }

    public static void sort(Comparable[] a) {
        Comparable[] aux = new Comparable[a.length];
        sort(a, aux, 0, a.length);
    }
}
```

Merging

Combine two pre-sorted lists into a sorted whole.

How to merge efficiently? Use an auxiliary array.

```java
for (int k = l; k < r; k++) aux[k] = a[k];
int i = l, j = m;
for (int k = l; k < r; k++)
    if (i >= m) a[k] = aux[j++];
    else if (j >= r) a[k] = aux[i++];
    else if (less(aux[j], aux[i])) a[k] = aux[j++];
    else a[k] = aux[i++];
```

Mergesort Analysis: Memory

Q. How much memory does mergesort require?
   - Original input array = N.
   - Auxiliary array for merging = N.
   - Local variables: constant.
   - Function call stack: \( \log_2 N \).
   - Total = \( 2N + O(\log N) \).

Q. How much memory do other sorting algorithms require?
   - \( N + O(1) \) for insertion sort and selection sort.
   - In-place = \( N + O(\log N) \).

Challenge for the bored. In-place merge. [Kronrud, 1969]
Mergesort Analysis: Running Time

**Def.** $T(N) =$ number of comparisons to mergesort an input of size $N$.

**Mergesort recurrence.**

$$T(N) = \begin{cases} 
0 & \text{if } N=1 \\
\frac{T\left(\lceil N/2 \rceil \right)}{2} + \frac{T\left(\lfloor N/2 \rfloor \right)}{2} + \frac{N}{2} & \text{if } N \neq 1 
\end{cases}$$

**Solution.** $T(N) = O(N \log_2 N)$.

- **Note:** same number of comparisons for any input of size $N$.
- We prove $T(N) = N \log_2 N$ when $N$ is a power of 2, and = instead of $\leq$.

**Proof by Induction**

**Claim.** If $T(N)$ satisfies this recurrence, then $T(N) = N \log_2 N$.

\[ T(N) = \begin{cases} 
0 & \text{if } N=1 \\
\frac{2T(N/2)}{2} + \frac{N}{2} & \text{if } N \neq 1 
\end{cases} \]

**Pf.** (by induction on $n$)

- **Base case:** $n = 1$.
- **Inductive hypothesis:** $T(n) = n \log_2 n$.
- **Goal:** show that $T(2n) = 2n \log_2 (2n)$.

\[
T(2n) = 2T(n) + 2n = 2n \log_2 n + 2n = 2(n \log_2 (2n) - 1) + 2n = 2n \log_2 (2n)
\]

Mergesort Analysis: Running Time

**Def.** $T(N) =$ number of comparisons to mergesort an input of size $N$.

**Mergesort recurrence.**

$$T(N) = \begin{cases} 
0 & \text{if } N=1 \\
\frac{T\left(\lceil N/2 \rceil \right)}{2} + \frac{T\left(\lfloor N/2 \rfloor \right)}{2} + \frac{N}{2} & \text{if } N \neq 1 
\end{cases}$$

**Solution.** $T(N) = O(N \log_2 N)$.

- **Note:** same number of comparisons for any input of size $N$.
- We prove $T(N) = N \log_2 N$ when $N$ is a power of 2, and = instead of $\leq$.

**Proof by Induction**

**Claim.** If $T(N)$ satisfies this recurrence, then $T(N) = N \log_2 N$.

\[ T(N) = \begin{cases} 
0 & \text{if } N=1 \\
\frac{2T(N/2)}{2} + \frac{N}{2} & \text{if } N \neq 1 
\end{cases} \]

**Pf.** (by induction on $n$)

- **Base case:** $n = 1$.
- **Inductive hypothesis:** $T(n) = n \log_2 n$.
- **Goal:** show that $T(2n) = 2n \log_2 (2n)$.

\[
T(2n) = 2T(n) + 2n = 2n \log_2 n + 2n = 2(n \log_2 (2n) - 1) + 2n = 2n \log_2 (2n)
\]

Mergesort Analysis: Running Time

**Def.** $T(N) =$ number of comparisons to mergesort an input of size $N$.

**Mergesort recurrence.**

$$T(N) = \begin{cases} 
0 & \text{if } N=1 \\
\frac{T\left(\lceil N/2 \rceil \right)}{2} + \frac{T\left(\lfloor N/2 \rfloor \right)}{2} + \frac{N}{2} & \text{if } N \neq 1 
\end{cases}$$

**Solution.** $T(N) = O(N \log_2 N)$.

- **Note:** same number of comparisons for any input of size $N$.
- We prove $T(N) = N \log_2 N$ when $N$ is a power of 2, and = instead of $\leq$.

**Proof by Induction**

**Claim.** If $T(N)$ satisfies this recurrence, then $T(N) = N \log_2 N$.

\[ T(N) = \begin{cases} 
0 & \text{if } N=1 \\
\frac{2T(N/2)}{2} + \frac{N}{2} & \text{if } N \neq 1 
\end{cases} \]

**Pf.** (by induction on $n$)

- **Base case:** $n = 1$.
- **Inductive hypothesis:** $T(n) = n \log_2 n$.
- **Goal:** show that $T(2n) = 2n \log_2 (2n)$.

\[
T(2n) = 2T(n) + 2n = 2n \log_2 n + 2n = 2(n \log_2 (2n) - 1) + 2n = 2n \log_2 (2n)
\]

Mergesort Analysis: Running Time

**Def.** $T(N) =$ number of comparisons to mergesort an input of size $N$.

**Mergesort recurrence.**

$$T(N) = \begin{cases} 
0 & \text{if } N=1 \\
\frac{T\left(\lceil N/2 \rceil \right)}{2} + \frac{T\left(\lfloor N/2 \rfloor \right)}{2} + \frac{N}{2} & \text{if } N \neq 1 
\end{cases}$$

**Solution.** $T(N) = O(N \log_2 N)$.

- **Note:** same number of comparisons for any input of size $N$.
- We prove $T(N) = N \log_2 N$ when $N$ is a power of 2, and = instead of $\leq$.

**Proof by Induction**

**Claim.** If $T(N)$ satisfies this recurrence, then $T(N) = N \log_2 N$.

\[ T(N) = \begin{cases} 
0 & \text{if } N=1 \\
\frac{2T(N/2)}{2} + \frac{N}{2} & \text{if } N \neq 1 
\end{cases} \]

**Pf.** (by induction on $n$)

- **Base case:** $n = 1$.
- **Inductive hypothesis:** $T(n) = n \log_2 n$.
- **Goal:** show that $T(2n) = 2n \log_2 (2n)$.

\[
T(2n) = 2T(n) + 2n = 2n \log_2 n + 2n = 2(n \log_2 (2n) - 1) + 2n = 2n \log_2 (2n)
\]

Mergesort Analysis: Running Time

**Def.** $T(N) =$ number of comparisons to mergesort an input of size $N$.

**Mergesort recurrence.**

$$T(N) = \begin{cases} 
0 & \text{if } N=1 \\
\frac{T\left(\lceil N/2 \rceil \right)}{2} + \frac{T\left(\lfloor N/2 \rfloor \right)}{2} + \frac{N}{2} & \text{if } N \neq 1 
\end{cases}$$

**Solution.** $T(N) = O(N \log_2 N)$.

- **Note:** same number of comparisons for any input of size $N$.
- We prove $T(N) = N \log_2 N$ when $N$ is a power of 2, and = instead of $\leq$.

**Proof by Induction**

**Claim.** If $T(N)$ satisfies this recurrence, then $T(N) = N \log_2 N$.

\[ T(N) = \begin{cases} 
0 & \text{if } N=1 \\
\frac{2T(N/2)}{2} + \frac{N}{2} & \text{if } N \neq 1 
\end{cases} \]

**Pf.** (by induction on $n$)

- **Base case:** $n = 1$.
- **Inductive hypothesis:** $T(n) = n \log_2 n$.
- **Goal:** show that $T(2n) = 2n \log_2 (2n)$.

\[
T(2n) = 2T(n) + 2n = 2n \log_2 n + 2n = 2(n \log_2 (2n) - 1) + 2n = 2n \log_2 (2n)
\]
Sorting Analysis Summary

Running time estimates:
- Home pc executes $10^8$ comparisons/second.
- Supercomputer executes $10^{12}$ comparisons/second.

<table>
<thead>
<tr>
<th></th>
<th>Insertion Sort (N²)</th>
<th>Mergesort (N log N)</th>
</tr>
</thead>
<tbody>
<tr>
<td>computer</td>
<td>thousand million</td>
<td>thousand million</td>
</tr>
<tr>
<td>home</td>
<td>instant</td>
<td>instant</td>
</tr>
<tr>
<td>super</td>
<td>2.8 hours 317 years</td>
<td>1 sec 18 min</td>
</tr>
<tr>
<td></td>
<td>instant 1 second 1.6 weeks</td>
<td>instant instant</td>
</tr>
</tbody>
</table>

Lesson 1. Good algorithms are better than supercomputers.

Quicksort

Quicksort.
- Shuffle the array.
- Partition array so that:
  - element $a[i]$ is in its final place for some $i$
  - no larger element to the left of $i$
  - no smaller element to the right of $i$
- Sort each piece recursively.

Q. How do we partition in-place efficiently?

Quicksort Partitioning
Quick sort: Java Implementation

private static int partition(Comparable[] a, int l, int r) {
    int i = l - 1;
    for (int j = l; j < r; j++) {
        if (less(a[j], a[r])) {
            i++;
            exch(a, i, j);
        }
    }
    exch(a, i + 1, r);
    return i;
}

public class Quick {
    public static void sort(Comparable[] a) {
        if (r <= l) return;
        int m = partition(a, l, r);
        sort(a, l, m-1);
        sort(a, m+1, r);
    }
}

Quick sort Example

Using a spare array makes partitioning easier, but is not worth the cost.

Partitioning in-place. Using a spare array makes partitioning easier, but is not worth the cost.

Terminating the loop. Testing whether the pointers cross is a bit trickier than it might seem.

Staying in bounds. The (i == r) test is redundant, but the (j == l) test is not.

Preserving randomness. Shuffling is key for performance guarantee.

Equal keys. When duplicates are present, it is (counter-intuitively) best to stop on elements equal to partitioning element.
**Worst case.** Number of comparisons is quadratic.
- \( N + (N-1) + (N-2) + \ldots + 1 = N(N+1)/2 \)
- More likely that your computer is struck by lightning.

**Caveat.** Many textbook implementations go quadratic if input:
- Is sorted.
- Is reverse sorted.
- Has many duplicates.

---

**Average case running time.**
- Roughly \( 2N \ln N \) comparisons.
- Assumption: file is randomly shuffled.

**Remarks.**
- 39% more comparisons than mergesort.
- Faster than mergesort in practice because of lower cost of other high-frequency instructions.
- Caveat: many textbook implementations have best case \( N^2 \) if duplicates, even if randomized!

---

**Theorem.** The average number of comparisons \( C_N \) to quicksort a random file of \( N \) elements is about \( 2N \ln N \).

\[
C_N = N + 1 + \frac{1}{N} \sum_{k=1}^{N-1} (C_k + C_{N-k})
\]

\[
= N + 1 + \frac{2}{N} \sum_{k=1}^{N-1} C_{k-1}
\]

Multiply both sides by \( N \) and subtract the same formula for \( N-1 \):
\[
NC_N - (N-1)C_{N-1} = N(N+1) - (N-1)N + 2C_{N-1}
\]

Simplify to:
\[
NC_N = (N+1)C_{N-1} + 2N
\]

Divide both sides by \( N(N+1) \) to get a telescoping sum:
\[
\frac{C_N}{N+1} = \frac{C_{N-1}}{N} + \frac{2}{N+1}
\]

\[
= \frac{C_{N-2}}{N-1} + \frac{2}{N} + \frac{2}{N+1}
\]

\[
= \frac{C_{N-3}}{N-2} + \frac{2}{N-1} + \frac{2}{N} + \frac{2}{N+1}
\]

\[
= \vdots
\]

\[
= \frac{C_2}{3} + \sum_{k=3}^{N} \frac{2}{k+1}
\]

Approximate the exact answer by an integral:
\[
\frac{C_N}{N+1} = \sum_{k=1}^{N} \frac{1}{k} = \int_{1}^{N} \frac{1}{x} = 2 \ln N
\]

Finally, the desired result:
\[
C_N = 2(N+1) \ln N = 1.39N \log_2 N
\]
Duplicate Keys

Equal keys: omnipresent in applications when purpose of sort is to bring records with equal keys together.
- Sort population by age.
- Finding collinear points.
- Remove duplicates from mailing list.
- Sort job applicants by college attended.

Typical application.
- Huge file.
- Small number of key values.
- Randomized 3-way quicksort is LINEAR time. (Try it!)

Theorem. Quicksort with 3-way partitioning is optimal.

Pf. Ties cost to entropy. Beyond scope of 226.

3-way partitioning. Partition elements into 3 parts:
- Elements between i and j equal to partition element v.
- No larger elements to left of i.
- No smaller elements to right of j.

3-Way Partitioning

Dutch national flag problem.
- Not done in practical sorts before mid-1990s.
- Incorporated into Java system sort, C qsort.

Lesson 1. Good algorithms are better than supercomputers.
Lesson 2. Great algorithms are better than good ones.
Selection

Selection. Find the kth largest element.
- Min: \( k = 1 \).
- Max: \( k = N \).
- Median: \( k = N/2 \).

Easy. Min or max with \( O(N) \) comparisons; median with \( O(N \log N) \).

Challenge. \( O(N) \) comparisons for any \( k \).
Quick-Select Analysis

Property C. Quick-select takes linear time on average.
- Intuitively, each partitioning step roughly splits array in half.
- \( N + \frac{N}{2} + \frac{N}{4} + \ldots < 2N \) comparisons.
- Formal analysis similar to quicksort analysis proves the average number of comparisons is
  \[
  2N + k \ln \left( \frac{N}{k} \right) + \left( N - k \right) \ln \left( \frac{N}{N-k} \right)
  \]
  Ex: \( (2 + 2 \ln 2) N \) comparisons to find the median

Worst-case. The worst-case is \( \Omega(N^2) \) comparisons, but as with quicksort, the random shuffle makes this case extremely unlikely.