Lecture 3:
- Algorithm Complexity Analysis (cont)

Announcements
- Problem Set 1: due Tues, Sept 13
- Quiz 1: Tues, Sept 13

---

**Merge Sort (Review)**

**Algorithm:** MergeSort

**Input:** Array $A$ of size $n$, start index, and end index ($0 \leq \text{start,end} < n$).

**Result:** Elements of $A$ in ascending order.

```plaintext
1 begin
2    if start < end then
3        mid ← (start + end) / 2
4        MergeSort($A$, start, mid)
5        MergeSort($A$, mid + 1, end)
6        Merge($A$, start, mid, end)
```

**Basic idea:** A classic recursive “divide and conquer” approach
- “divide” array into two halves
- sort each half (recursive step)
- merge the sorted halves into sorted array
Merge Sort (cont)

Algorithm: Merge
Input: Array $A$ of size $n$ with start, mid, and end indices such that both $A[start .. mid]$ and $A[mid+1 .. end]$ are in ascending order.

1 begin
2 $T$ is a new (temporary) array of size $m = (end - start) + 1$
3 $i \leftarrow 0$, $j_1 \leftarrow start$, $j_2 \leftarrow mid + 1$
4 while $j_1 \leq mid$ and $j_2 \leq end$ do
6     else $T[i++] \leftarrow A[j_2++]$
7 while $j_1 \leq mid$ do $T[i++] \leftarrow A[j_1++]$
8 while $j_2 \leq end$ do $T[i++] \leftarrow A[j_2++]$
9 for $i \leftarrow 0$ to (end-start) do $A[start + i] \leftarrow T[i]$

Q: What is the best case, worst case, and (Big-$\Theta$) cost for Merge?

- **best case**: already sorted, **worst case**: “interleaved”
- assuming $n$ total elements ... $\Theta(n)$

Merge Sort Complexity Analysis: Informal

- At each “merge step” we do $\Theta(n)$ amount of work (from merge)
- Given a list of size $n$, how many “merge steps” are there?

- Thus, $\Theta(\log n)$ merge steps each costing $\Theta(n)$
- This means mergesort is $\Theta(n \log n)$!
Merge Sort Complexity Analysis: Recurrence Relations

(1). We have the following recurrence for MergeSort

\[ T(n) = \begin{cases} 
    c_1 & n = 0 \text{ or } n = 1 \\
    c_1 + c_2 + T(\frac{n}{2}) + T(\frac{n}{2}) + c_3 \cdot n + c_4 & n > 1 
\end{cases} \]

where each \( c_i \) is a constant such that:
- \( c_1 \) is the start < end comparison cost
- \( c_2 \) is the mid calculation cost
- \( c_3 \cdot n + c_4 \) is the merge step cost

(2). Can simplify and generalize the non-recurrence terms

\[ T(n) = \begin{cases} 
    \Theta(1) & n = 0 \text{ or } n = 1 \\
    2 \cdot T(\frac{n}{2}) + \Theta(n) & n > 1 
\end{cases} \]

(3). Solving the recurrence relation gives the asymptotic complexity:
- we will “solve” the recurrence using the master method

The Master Method (Divide and Conquer)

(4) A “standard recurrence” has the general form:

\[ T(n) = O(1) \quad \text{... for small } n \]

\[ T(n) \leq a \cdot T(\frac{n}{b}) + O(n^d) \quad \text{... otherwise} \]

where:
- \( a \) is the number of recursive calls
- \( b \) is size of division (in divide and conquer)
- \( \frac{n}{b} \) is assumed to be either \([ \frac{n}{b} ]\) or \([ \frac{n}{b} ]\)
- \( d \) is exponent in the combine step

(5) If \( T(n) \) is defined by a standard recurrence, then:

\[ T(n) \text{ is} \begin{cases} 
    O(n^d \log n) & \text{if } a = b^d \\
    O(n^d) & \text{if } a < b^d \\
    O(n^{\log_b a}) & \text{if } a > b^d 
\end{cases} \]
Q: What is the result of the Master Method for MergeSort?

Recall:

\[ T(n) = O(1) \] (for small \( n \))
\[ T(n) = 2T(\frac{n}{2}) + O(n) \] (for large \( n \))

Thus:

\[ a = 2, \ b = 2, \ d = 1 \]

And so: \( T(n) \) is \( O(n \log n) \)

Q: What about for Binary Search?

As a standard recurrence relation:

\[ T(n) = O(1) \] (for small \( n \))
\[ T(n) = T(\frac{n}{2}) + O(1) \] (for large \( n \))

Thus:

\[ a = 1, \ b = 2, \ d = 0 \]

And so: \( T(n) \) is \( O(\log n) \)

The Master Method (Reduce and Conquer)

Can use a similar approach for "iterative" algorithms:

- Define as a standard recurrence relation (i.e., as a recursive \( T \) function)
- Apply the reduce-and-conquer version of the master method

Standard form for reduce-and-conquer algorithms:

\[ T(n) = O(1) \] ... for small \( n \)
\[ T(n) \leq a \cdot T(n - b) + O(n^d) \] ... reduce \( n \) by \( b \) on each iteration

Master Method for reduce-and-conquer algorithms:

\[
T(n) \text{ is } \begin{cases} 
O(n^{d+1}) & \text{if } a = 1 \\
O(n^d a^{n/b}) & \text{if } a > 1 \\
O(n^d) & \text{otherwise}
\end{cases}
\]

Note: The last case is only provided for completeness ...
Q: What is the result of the Master Method for Linear Search?

As a reduce-and-conquer recurrence relation:

\[
T(n) = O(1) \quad \text{(for } n = 0 \text{ and } n = 1) \\
T(n) = T(n - 1) + O(1) \quad \text{(for } n > 1) \\
\]

i.e., each “iteration”, check only first element (of smaller list)

Thus:

\[a = 1, \ b = 1, \ d = 0\]

And so: \(T(n) \text{ is } O(n)\)