Bounding sums. Recall that, for example, $\sum_{i=1}^n i^2$ denotes the sum $1^2+2^2+\cdots+n^2$. You may have previously learned *closed forms* for some sums, such as $\sum_{i=1}^n i = n(n+1)/2$. Unfortunately, for typical sums of interest, closed forms are difficult to find or simply don't exist. Fortunately, when we are analyzing asymptotic running times, we don't care about lower-order terms or constant factors, so we don't usually need exact closed forms for the sums that arise. Instead, for example, we only need to know that $\sum_{i=1}^n i = \Theta(n^2)$.

Here are some handy rules for quickly getting such bounds.

Observation 1 (naive upper bound)**.** *Any sum is at most the number of terms times the maximum term.*

Examples:

$$\begin{aligned}
\textstyle\sum_{i=1}^n i \;&\le\; n\cdot \max_{i=1}^{n} i \;=\; n\cdot n \;=\; O(n^2)\\
\textstyle\sum_{i=1}^n i^2 \;&\le\; n\cdot \max_{i=1}^{n} i^2 \;=\; n\cdot n^2 \;=\; O(n^3)\\
\textstyle\sum_{i=1}^n \log i \;&\le\; n\cdot \max_{i=1}^{n} \log i \;=\; n\log n \;=\; O(n\log n)
\end{aligned}$$

Observation 2 (naive lower bound)**.** *Any sum with non-negative terms is at least half the number of terms times the median term.*

Examples:

$$\begin{aligned}
\textstyle\sum_{i=1}^n i \;&\ge\; (n/2)\cdot(n/2) \;=\; \Omega(n^2)\\
\textstyle\sum_{i=1}^n i^2 \;&\ge\; (n/2)\cdot(n/2)^2 \;=\; \Omega(n^3)\\
\textstyle\sum_{i=1}^n \log i \;&\ge\; (n/2)\cdot\log(n/2) \;=\; \Omega(n\log n)
\end{aligned}$$

For each of the three examples above, the naive upper and lower bounds are enough to determine the value of the sum up to constant factors. For example, $\sum_{i=1}^n \log i = \Theta(n\log n)$. This will work for any sum whose terms don't increase (or decrease) too fast. Specifically, any sum where the median term is within a constant factor of the maximum term.

However, it won't work for all sums. For example, try $\sum_{i=1}^n 2^i$. The naive upper bound gives $\sum_{i=1}^n 2^i = O(n2^n)$. The naive lower bound gives $\sum_{i=1}^n 2^i = \Omega(2^{n/2})$. But these bounds are very far apart! Indeed their ratio $n2^n/2^{n/2} = n2^{n/2}$ is quite large. Neither bound is tight in this case, as $\sum_{i=1}^n 2^i = \Theta(2^n)$. To handle sums like this, here is one more general rule.

Definition 1. *A sum $\sum_{i=1}^n \alpha_i$ is* geometric *if, for some constants $c>1$ and i_0 (indep. of n) either*
(i) for each $i \ge i_0$ the ith term is at least c times the previous term (that is, $\alpha_i \ge c\cdot\alpha_{i-1}$), or
(ii) for each $i\ge i_0$ the ith term is at most the previous term divided by c (that is, $\alpha_i \le \alpha_{i-1}/c$).

For example, $\sum_{i=1}^n 2^i$ is geometric, because the ratio of successive terms $2^i/2^{i-1}$ is at least two. Also, $\sum_{i=1}^n 2^i/i$ is also geometric: for $i\ge 3$ the ratio of successive terms is $(2^i/i)/(2^{i-1}/(i-1))$ $= 2(i-1)/i \ge 4/3$. The sum $\sum_{i=1}^n i^2+1$ is not geometric, because the ratio of successive terms $((i+1)^2+1)/(i^2+1)$ gets arbitrarily close to 1 for large i.

Observation 3 (geometric sums)**.** *Any geometric sum is proportional to its largest term.*

Some examples:

$$\begin{aligned}
\textstyle\sum_{i=0}^n 2^i &= \Theta(\max_{i=0}^n 2^i) = \Theta(2^n).\\
\textstyle\sum_{i=1}^n 2^i/i &= \Theta(\max_{i=1}^n 2^i/i) = \Theta(2^n/n).\\
\textstyle\sum_{i=0}^\infty i^2/2^i &= \Theta(\max_{i=0}^\infty i^2/2^i) = \Theta(1).
\end{aligned}$$

Recursions trees. Next we discuss how to analyze the run times of recursive algorithms.

We use mergesort as an example.

```
mergesort(A[1..n])
if n = 1, return
set m = ⌊n/2⌋
mergesort(A[1..m])                  (sort the first n/2 elements)
mergesort(A[m+1..n])                (sort the last n/2 elements)
merge(A[1..n], m)                   (merge the two sorted halves, in time Θ(n))
```

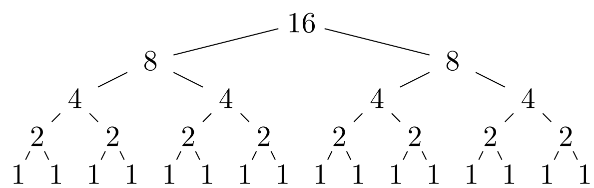
The merge procedure runs in linear time. We don't give the code for it.

Let $T(n)$ denote the worst-case running time of mergesort on any array of size n. Then $T(1) = \Theta(1)$. For $n \geq 1$, by inspection of mergesort and using that merge takes linear time, we have the recurrence relation

$$T(n) = 2\,T(n/2) + n.$$

(For now, we assume n is a power of 2, so $n/2$ is an integer.)

On the left below is the recursion tree for mergesort on an array of size $n = 16$. Each node represents one call to mergesort, and is labeled by the size of the subarray that the call sorts.



level	# of subprobs	subprob. size	work per subprob	work in level
0	1	16	16	16
1	2	8	8	16
2	4	4	4	16
3	8	2	2	16
4	16	1	1	16
			total:	5×16

As illustrated in the table to the right above, if the input array size n is a power of 2, there are $1+\log_2 n$ levels. For each $i \in \{0, \ldots, \log_2 n\}$, level i has 2^i subproblems, each of size $n/2^i$. The work associated with a subproblem of size $n/2^i$ is $\Theta(n/2^i)$, so the total work at level i is $\Theta(2^i \times n/2^i) = \Theta(n)$. Summing over the $\log n$ levels, the total work is $\Theta(n \log n)$. Hence, $T(n) = \Theta(n \log n)$. Note that the work is balanced evenly across the levels.

If n is not a power of 2, then there are $1 + \lceil \log_2 n \rceil$ levels. In each level i except the last, there are 2^i subproblems, each of size between $n/2^{i+1}$ and $n/2^i$. So, by a similar calculation, the total work is also $\Theta(n \log n)$ in this case.

Variations. Consider the recurrence relation $T(n) = \mathbf{3} \cdot T(n/2) + n$. Then we get

$$\begin{aligned} T(n) &= \sum_{i=0}^{\text{max level}} (\text{\# subprobs on level } i) \times (\text{work per subprob. on level } i) \\ &= \sum_{i=0}^{\log_2 n} 3^i (n/2^i) \;=\; n \sum_{i=0}^{\log_2 n} (3/2)^i \;=\; \Theta(n\,(3/2)^{\log_2 n}). \end{aligned}$$

The last step follows because the sum is geometric.

Here *the total work is dominated by the work done at the leaves.* Using the identity $a^{\log b} = b^{\log a}$, we can simplify this to $T(n) = \Theta(n \cdot n^{\log_2 3/2}) = \Theta(n^{\log_2 3}) = \Theta(n^{1.58\ldots})$.

Consider the recurrence relation $T(n) = 2T(n/2) + n^2$. Then we get

$$T(n) = \sum_{i=0}^{\log_2 n} 3^i (n/2^i)^2 = n^2 \sum_{i=0}^{\log_2 n} (3/4)^i = \Theta(n^2).$$

The last step follows because the sum is geometric. Here *the total work is dominated by the work at the root.*

These three cases occur often: either (a) the work is balanced across the levels, or the sum is geometric, so (b) the work at the leaves dominates, or (c) the work at the root dominates. A particularly important special case of the latter case (when the work at the root dominates) is when *the algorithm does linear work, then recurses on subproblems whose combined size is a constant factor less than the size of the given problem.*

For example, consider the recurrence relation $T(n) = aT(n/b) + n$, for some $a < b$. That is, the algorithm does linear work, then recurses on a problems, each of size n/b, so having total size $an/b = (a/b)n$ where $a/b < 1$. Then we get

$$T(n) = \sum_{i=0}^{\log_b n} a^i (n/b^i) = n \sum_{i=0}^{\log_b n} (a/b)^i = \Theta(n).$$

The last step follows because $a < b$, so the sum is geometric and dominated by the first term.

External resources on analyzing recursive algorithms

- CLRS Chapter 4. Dasgupta et al. Chapter 2.2. Kleinberg & Tardos Chapter 5.
- Jeff Edmond's Appendix II.

  http://jeffe.cs.illinois.edu/teaching/algorithms/notes/99-recurrences.pdf
- MIT lecture videos
  - https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-006-introduction-to-algorithms-fall-2011/lecture-videos/lecture-3-insertion-sort-merge-sort/