Bounding Facts

Prof. Dan Barrish-Flood
Algorithms
More Bounding Facts

If \( f(n) = o(g(n)) \), then \( f(n) \neq \Theta(g(n)) \)

Proof:

Assume \( c_1 g(n) \leq f(n) \), for \( n \geq n_0 \)

Divide by \( g(n) \):

\[
c_1 \leq \frac{f(n)}{g(n)}, \quad n \geq n_0
\]

But (by definition of "\(o\")

\[
\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0
\]
More Bounding Facts (continued)

So eventually (for $n$ large enough)

$$\frac{f(n)}{g(n)} < c_1. \text{ Contradiction!}$$
Summations

\[ \sum_{i=0}^{n} a_i = a_0 + a_1 + \ldots + a_n \]

Linearity:

\[ \sum_i (ca_i) = c \sum_i a_i \]
\[ \sum_i (a_i + b_i) = \sum_i a_i + \sum_i b_i \]
\[ \sum_i (ca_i + b_i) = c \sum_i a_i + \sum_i b_i \]

Example:

\[ \sum_i (5i^2 + 2i) = \]
\[ \sum_i 5i^2 + \sum_i 2i = \]
\[ 5 \sum_i i^2 + 2 \sum_i i \]
Common Summations

\[ \sum_{i=1}^{n} i = 1 + 2 + \ldots + n = \frac{n(n + 1)}{2} = \frac{(n^2 + n)}{2} = \Theta(n^2) \]

\[ \sum_{i=0}^{n} 2^i = 1 + 2 + 4 \ldots + 2^n = 2^{n+1} - 1 \]

Proof: Let \( S = \sum_{i=0}^{n} 2^i \). Then \( 2S = \sum_{i=1}^{n+1} 2^i \)

\[ 2S - S = S = 2^{n+1} - 1 \]
In general:
\[ \sum_{i=0}^{n} r^i = \frac{r^{n+1} - 1}{r - 1} \]
and noting that if \(|r| < 1\), then
\[ \sum_{i=0}^{\infty} r^i = \frac{1}{1 - r} \]

Proof:
\[ r \sum_{i=0}^{n} r^i = r + r^2 + r^n + r^{n+1} \]

Subtract
\[ \sum_{i=0}^{n} r^i = 1 + r + r^2 + \ldots + r^n \]

\[ (r - 1) \sum_{i=0}^{n} r^i = -1 + r^{n+1} \]

Divide by \((r - 1)\)

So...
\[ \sum_{i=0}^{n} r^i = \frac{r^{n+1} - 1}{r - 1} \]
Running Times for Some Simple Iterative Algorithms

CLEAR-MATRIX(A[1 \ldots n,1\ldots n])

1 for i ← 1 to n
2 do for j ← 1 to n
3 do A[i,j]=0

\[
\begin{bmatrix}
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
\end{bmatrix}
\]

\[
T(n) = \Theta(n) + \Theta(n^2) = \Theta(n^2)
\]
IDENTITY-MATRIX(A[1 \ldots n,1\ldots n])

1 CLEAR-MATRIX(A)
2 for i ← 1 to n do
3 do A[i,i]=1

\[
T(n) = \Theta(n) + \Theta(n^2) = \Theta(n^2)
\]

same asymptotic running time as CLEAR-MATRIX
CLEAR-LOWER-TRIANGLE(A[1…n,1…n])
(including diagonal)
1 for i ← 1 to n
2 do for j ← 1 to i
3 do A[i,j]=0

Line 3 happens $\sum_{i=1}^{n} i$ times.

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \Theta(n^2)$$
CLEAR-UPPER-TRIANGLE(A[1\ldots n,1\ldots n])  
(including diagonal)

1  for  i ← 1 to n
2     do for  j ← i to n
3        do A[i,j]=0

Line 3 happens \( \sum_{i=1}^{n} (n - i + 1) \) times.

Let \( k = n - i + 1 \). Then we get

\[
\sum_{k=1}^{n} k = \frac{n(n+1)}{2} = \Theta(n^2)
\]
AVERAGE-NEIGHBORS(A[1…n])

1  for i ← 2 to n-1 do
2       sum ← 0
3  for j ← i−1 to i+1 do
4       sum ← sum+A[j]
5  A[i] ← sum/3

1 8 3 2 5 1 6

4 4 1/3 3 1/3 2 2/3 4

T(n)=θ(n) LINEAR!
Mergesort

- To sort a list of n items:
  • sort the first half of the list
  • sort the second half
  • merge the sorted halves
- Sorting 1 or 0 items is trivial.
- Merging two sorted lists is easily done in linear ($\theta(n)$) time. (Just take the smallest item from each list and put it on the merged list.)
MERGESORT(A[1…n])
1  if n ≤ 1 then
2    return A
3  else
4    B ← MERGESORT(A[1 … ⌊n/2⌋])
5    C ← MERGESORT(A[⌊n/2⌋+1…n])
6    return MERGE(B,C)
Mergesort Example

The operation of mergesort on the array \(A = \langle 5, 2, 4, 6, 1, 3, 2, 6 \rangle\). The lengths of the sorted sequences being merged increases as the algorithm progresses from bottom to top.
The Call Tree for Mergesort

MS(n)

MS(n/2)                MS(n/2)

MS(n/4)   MS(n/4)   MS(n/4)  MS(n/4)

...MS(1) ...................................... MS(0)...........
What is the running time of Mergesort?
Let’s call it $T(n)$.

$$T(n) = \text{time taken by recursive calls} + \text{time to merge}$$

$$T(n) = 2T(n/2) + n$$

$T(1) = 1$

This is a \textit{recurrence equation}, or recurrence for short.
Solving Recurrences

Method 1: Know the answer.
  – Not recommended as a general technique.

Method 2: Recursion Trees
  – Good for intuition

Method 3: Telescoping + Transformations
  – Fairly general
Recursion Trees

1. Draw the call tree for the algorithm.
2. Label each node with the work done at that invocation only.
3. Determine the total work done at each level of the tree.
4. Sum the values at each level to get the total running time.
Recursive Factorial

FACT(n)
1 if n ≤ 1 then
2 return 1
3 else
4 return n*FACT(n-1)

T(1) = 1
T(n) = T(n-1) + 1, n ≥ 2

Call tree:
FACT(n)
FACT(n-1)
FACT(n-2)
FACT(1)
Recursion Tree for FACT

total work
at each level

levels

n

1

1

1

1

1

1

1

1

n
Recursion Tree for Mergesort

\[
n(\lceil \log n \rceil + 1) = n \lceil \log n \rceil + n = \Theta(n \log n)
\]
Telescoping

Consider

\[ T(n) = T(n-1) + 1, \text{ } n \geq 1 \]

\[ T(0) = 1 \]

(The recurrence for \text{FACT}(n).)

“world’s simplest recurrence”

We can solve it like this:

\[ T(n) - T(n-1) = 1 \]

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

\[ T(n-2) - T(n-3) = 1 \]

\[ \vdots \]

\[ T(1) - T(0) = 1 \]

\[ T(n) - T(0) = n \]

\[ T(n) = n+1 \]
Generalizing ...

The RHS’s can be different from each other - as long as they don’t contain $T$.

E.g. $T(n) = T(n - 1) + 2^n$, $n \geq 1$

$T(0) = 1$

$T(n) - T(n - 1) = 2^n$

$T(n - 1) - T(n - 2) = 2^{n-1}$

$\vdots$

$T(1) - T(0) = 2^1$

$T(n) - T(0) = \sum_{i=1}^{n} 2^i$

$T(n) = \sum_{i=0}^{n} 2^i = 2^{n+1} - 1$

Check
Also...

The LHS doesn’t have to be $T(n)$. All we need is that
\[ \text{expr}(n)-\text{expr}(n-1) = \ldots \]

E.g. $nT(n) = (n-1)T(n-1) + 1$, $T(1) = 1$

\[
\begin{align*}
  nT(n) - (n-1)T(n-1) &= 1 \\
  (n-1)T(n-1) - (n-2)T(n-2) &= 1 \\
  &\vdots \\
  2T(2) - T(1) &= 1 \\
  \hline
  nT(n) - T(1) &= n-1
\end{align*}
\]

$nT(n) = n$, $T(n) = 1$
Finally...

It doesn’t matter whether the “base case” is $T(0)$, $T(1)$, $T(2)$, etc. - as long as it’s some constant.

Our goal is to get all recurrences into telescoping form.
Domain Transformations

Consider: \( T(n) = T(n/2) + 1 \), \( n \geq 2 \)

\[ T(1) = 1 \quad \text{(The recurrence for binary search)} \]

Not in the form we want, which is \( T(n) = T(n-1) + \ldots \)

But let’s do the following:

\[ n = 2^k \quad (k = \log_2 n) \]

\[ S(k) = T(2^k) \]

Now:

\[ T(n) = T(2^k) = S(k) \]
\[ T(n/2) = T(2^{k-1}) = S(k - 1) \]
\[ T(1) = T(2^0) = S(0) \]
Our recurrence

\[ T(n) = T(n/2) + 1, \ n \geq 2 \]

\[ T(1) = 1 \]

has become

\[ S(k) = S(k-1) + 1, \ k \geq 1 \]

\[ S(0) = 1 \]

which telescopes, giving \( S(k) = k + 1. \)

Back-substituting:

\[ T(n) = S(k) = k + 1 = \log n + 1 \]

(since we defined \( k = \log n. \))
Another recurrence:

\[ T(n) = T(n/2) + n, \quad T(1) = 1 \]

Letting \( n = 2^k \), \( S(k) = T(2^k) \)

\[ S(k) = S(k - 1) + 2^k \]

We've solved this on slide #23 - it's \( 2^{k+1} - 1 \).

Since \( n = 2^k \), we get (noting \( 2^{k+1} = 2 \times 2^k \))

\[ S(k) = T(n) = 2n - 1 \]
Range Transformations

\[ T(n) = 2T(n-1) + 1, \quad T(1) = 1. \]

(recurrence for “Towers of Hanoi” puzzle.)

The “2” multiplying \( T(n-1) \) is the problem.

So let’s multiply through by \( 2^{-n} \).

(Trust me.) We call this a *summing factor*.

\[
2^{-n} T(n) = 2^{-n} \times 2 \times T(n-1) + 2^{-n} \\
= 2^{-n+1} T(n-1) + 2^{-n} \\
= 2^{-(n-1)} T(n-1) + 2^{-n}
\]

It’s in telescoping form!
Let $S(n) = 2^{-n} T(n)$.

Then

$$2^{-n} T(n) = 2^{-(n-1)} T(n-1) + 2^{-n}$$

becomes

$$S(n) = S(n-1) + 2^{-n}$$

and $T(1) = 1$ becomes $S(1) = 1/2$.

Telescoping, we get

$$S(n) = \frac{1}{2^n} + \frac{1}{2^{n-1}} + \cdots + \frac{1}{2}$$

$$= \sum_{i=1}^{n} \left( \frac{1}{2} \right)^i$$
\[
\sum_{i=1}^{n} \left(\frac{1}{2}\right)^i = \sum_{i=0}^{n} \left(\frac{1}{2}\right)^i - 1
\]

\[
= \frac{\left(\frac{1}{2}\right)^{n+1} - 1}{\frac{1}{2} - 1} - 1 = \frac{\left(\frac{1}{2}\right)^n \times \frac{1}{2} - 1}{-\frac{1}{2}} - 1
\]

\[
= -\left(\frac{1}{2}\right)^n + 2 - 1 = 1 - \frac{1}{2^n}
\]

Since \( S(n) = 2^{-n} T(n) \), then

\[
T(n) = 2^n S(n)
\]

\[
= 2^n \left(1 - \frac{1}{2^n}\right)
\]

\[
= 2^n - 1.
\]
Now we are ready to tackle MERGESORT’s recurrence.

\[ T(n) = 2T(n/2) + n, \quad n > 1 \]
\[ T(1) = 1 \]

**First**, a domain transformation:

\[ n = 2^k, \quad S(k) = T(2^k) \text{ yields} \]
\[ S(k) = 2S(k - 1) + 2^k, \quad k \geq 1 \]
\[ S(0) = 1 \]

Now a range trans:

\[ R(k) = 2^{-k} S(k) \]

gives

\[ R(k) = R(k - 1) + 1, \quad k \geq 1 \quad \text{world’s simplest recurrence!} \]
\[ R(0) = 1 \]
We can solve that:

\[ R(k) = k + 1 \]

Back-substituting:

\[
S(k) = 2^k R(k), \quad \text{so}
\]

\[
S(k) = 2^k (k + 1)
\]

\[ k = \lg n, \quad \text{so} \]

\[
S(k) = T(n) = 2^{\lg n} (\lg n + 1)
\]

\[ = n(\lg n + 1) \]

So \( T(n) = \Theta(n \lg n) \).
The Telescope/Transformation Method: A Summary

1. Do domain transformations to deal with the argument to $T$.
   E.g. $T(n/2) \rightarrow S(k-1)$.

2. Do range transformations to handle multipliers of $T$.
   E.g. $2T(n-1) \rightarrow 2^{-(n-1)} T(n-1)$.

3. Telescope.

4. Check.