- Only 2 hours lecture tonight (yay!)
- But, hand back test at the end (boo!)
- Remarks—please submit form (email or otherwise) “pretty soon” after you get work back

\[ n \log n \quad n! \]

\[ n \log n \quad n^2 \]

\[ n \log n \quad \text{radix sort} \]

\[ 0(n) \]
mergeSort coded

Assume `merge(A, f, m, l)` is a linear in \( n = l - f + 1 \) algorithm to merge sorted arrays \( A[f..m] \) and \( A[m+1..l] \). You can use the techniques of 165 on the listing on page 68 to confirm this.

Use `merge` in the definition of `mergeSort`:

```python
def mergeSort(A, f, l):
    if (f == l): return
    else:
        m = (f + l) / 2
        mergeSort(A, f, m)
        mergeSort(A, m + 1, l)
        merge(A, f, m, l)
```

\[
T(n) = \begin{cases} 
1 & \text{if } n = 1 \\
T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + \Theta(n) & \text{otherwise}
\end{cases}
\]

\[n = f + l + 1\] — size of \( A \)
mergeSort counted

Count anything that doesn’t depend on $n$ as a constant:

$$T(n) = \begin{cases} 1, & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + dn, & \text{if } n > 1 \end{cases}$$

Try unwinding this in the special case where $n$ is a power of 2, say $n = 2^k$:

$$T(2^k) = 2 T(2^{k-1}) + d 2^k$$
$$= 2^2 T(2^{k-2}) + d 2^k + d 2^k$$
$$= 2^3 T(2^{k-3}) + 3 d 2^k$$
$$\vdots$$
$$= 2^k T(2^{k-k}) + k d 2^k$$
$$= n + d n \log n$$
\[ T(n) = \begin{cases} n & \text{if } n = 1 \\ \frac{T(n/2) + T(n/2) + d_n}{2} & \text{if } n > 1 \end{cases} \]

\[ P(k) : T(2^k) = 2^k + kd2^k \]

Claim: \( \forall k \in \mathbb{N}, P(k) \)

**Base case** 
\( P(0) \) says \( T(2^0) = T(1) = 1 = 2^0 + 0 \cdot d2^k \), so \( P(0) \) is true.

**Induction step** 
Assume \( P(k) \) holds for some generic \( k \in \mathbb{N} \). Then

\[ T(2^{k+1}) = T\left( \left\lfloor \frac{2^{k+1}}{2} \right\rfloor \right) + T\left( \left\lceil \frac{2^{k+1}}{2} \right\rceil \right) + d \cdot 2^{k+1} \]

\[ = 2T(2^k) + d \cdot 2^{k+1} \]

\[ = 2(2^k + kd2^k) + d \cdot 2^{k+1} \]

\[ = 2^{k+1} + kd2^{k+1} + d2^{k+1} = 2^{k+1} + (k+1)d2^{k+1} \]

So, \( P(k+1) \) is true, and \( \forall k \in \mathbb{N}, P(k) \Rightarrow P(k+1) \)

Conclude \( \forall k \in \mathbb{N}, P(k) \)
non-powers of 2?

Tactic:

First, solve for powers of 2

Then, show that $T(n)$ is monotonically non-decreasing

Then use monotonicity to solve between powers of 2
Proof: Complete induction.

Assume $n \in \mathbb{N}^+$. If $n = 1$, there are no counterexamples, so $m \in \mathbb{N}^+, m < 1$, so $P(1)$ is true. If $n = 2$, then $T(n) = T\left(\left\lceil \frac{2}{2} \right\rceil \right) + T\left(\left\lfloor \frac{2}{2} \right\rfloor \right) + 2d = 2 + 2d > T(1) = 1$, so $P(2)$ is true.

Otherwise $n \geq 3$, I must show that $T(n-1) \leq T(n)$.

(Otherwise covered by $P(n-1)$).

\[ T(n-1) = T\left(\left\lfloor \frac{n-1}{2} \right\rfloor \right) + T\left(\left\lceil \frac{n-1}{2} \right\rceil \right) + (n-1)d \]

\[ \leq T\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + T\left(\left\lceil \frac{n}{2} \right\rceil \right) + nd \]

\[ = T(n). \]

So, $\forall n \in \mathbb{N}^+, P(1) \ldots P(n-1) \Rightarrow P(n)$.

Conclude $\forall n \in \mathbb{N}^+, P(n)$.
Claim: \( \forall n \in \mathbb{N}, n > 1 \Rightarrow T(n) \leq X n \lg n \) (X will be determined during the proof).

Assume \( n \in \mathbb{N}, n > 1 \). Let \( y = \lg n \), so \( n = 2^y \), so

\[
2^{\frac{y}{y+1}} < n = 2^y \leq 2^{\frac{y}{y+1}}
\]

\[
T(n) = T(2^y) \leq T\left(2^{\frac{y}{y+1}}\right) \quad \text{# monotonicity}
\]

\[
= 2^{\frac{y}{y+1}} + d 2^{\frac{y}{y+1}} \frac{y}{y+1} \quad \text{# } y+1 > \frac{y}{y+1}
\]

\[
\leq 2^{y+1} + d 2^{y+1} \frac{y+1}{y+1} \quad \text{# } y = \lg n \geq 1, \ n \geq 2
\]

\[
= 2^{y+1} + d 2^{y+1} \frac{y+1}{y+1} \quad \text{# } \geq 1, \ n \geq 2
\]

\[
\leq 2 \cdot 2^y \cdot y + 2d \cdot 2^y \cdot y + 2d \cdot 2^y \cdot y
\]

\[
= 2 \cdot 2^y \cdot y + 4d \cdot 2^y \cdot y = (2 + 4d) 2^y \cdot y
\]

\[
= (2 + 4d) n \lg n
\]

\[ \hat{n} \]
mergeSort is an instance of a general strategy: divide up a problem into subproblems of the same type, and then combine the solutions to the subproblems. The “smallest” instances of the problem can be solved directly (for example, mergeSorting an array of size 1).

We can parameterize the different parts of this strategy. Suppose we break the problem into $b$ roughly equal pieces, that is $a_1$ pieces of size $\lceil n/b \rceil$ and $a_2$ pieces of size $\lfloor n/b \rfloor$. Suppose the number of steps to break the problem up, and then recombining the results is given by $dn^l$. Further, suppose that the number of steps required to solve the smallest instances of the problem, $1 \leq n < b$ is a constant $c$. Then the general form for the complexity of the problem is:

$$t(n) = \begin{cases} 
c, & \text{if } 1 \leq n < b \\
a_1 T(\lceil n/b \rceil) + a_2 T(\lfloor n/b \rfloor) + dn^l, & \text{if } n \geq b
\end{cases}$$
\[ T(n) = cT\left(\left\lceil \frac{n}{2} \right\rceil \right) + n^{1/2} \]

\[ t(n) = \begin{cases} 
  c, & \text{if } 1 \leq n < b \\
  a_1 T\left(\lfloor n/b \rfloor \right) + a_2 T\left(\lceil n/b \rceil \right) + dn^l, & \text{if } n \geq 1
\end{cases} \]

Again, we use the strategy (Course notes 87–89) of solving this equation for the special case when \( n = b^k \) for some natural number \( k \), and then extending this solution to natural numbers that aren’t powers of \( b \) by showing that \( T \) is non-decreasing. The result is the following asymptotic bound, where \( a = a_1 + a_2 \):

\[ T(n) \leq c_1 b^k \]

\[ T(n) \in \begin{cases} 
  \Theta(n^l), & \text{if } a < b^l \\
  \Theta(n^l \log_b n), & \text{if } a = b^l \\
  \Theta(n^{\log_b a}), & \text{if } a > b^l
\end{cases} \]

This result is the master theorem for divide-and-conquer algorithms.