Running times continued

- some running times are more difficult to analyze
- e.g., recursive algorithms

Example: **MergeSort**

Input: array of numbers
Output: sorted array of \( \uparrow \)

\[ 2 \ 7 \ 3 \ 6 \ \underline{1} \ 8 \ 5 \ 4 \]

Split in two halves
recursively sort left half
right half
merge the sorted halves

\[ 2 \ 3 \ 6 \ 7 \ \underline{1} \ 4 \ 5 \ 8 \]

\[ 1 \ 2 \ 3 \ 4 \ \ldots \]
### Merging two sorted lists

**Input:** Two arrays $A = \{a_1, a_2, \ldots, a_m\}$, $B = \{b_1, b_2, \ldots, b_n\}$, in increasing order

**Output:** Array $C$ containing $A \cup B$, in increasing order

**MERGE(A,B)**
1. $i=1; j=1; k=1$;
2. $a_{m+1} = \infty; b_{n+1} = \infty$;
3. while ($k \leq m+n$) do
4. if ($a_i < b_j$) then
5. $c_k = a_i; i++$;
6. else
7. $c_k = b_j; j++$;
8. $k++$;
9. RETURN $C=\{c_1, c_2, \ldots, c_{m+n}\}$

**Running time?** $O(m+n)$
Running times continued

Sorting

Input: An array $X = \{x_1, x_2, \ldots, x_n\}$
Output: $X$ sorted in increasing order

**MergeSort** - a divide-and-conquer algorithm

```
MERGESORT(X,n)  T(n)
1. if (n == 1) RETURN X
2. middle = n/2 (round down)
3. A = \{x_1, x_2, \ldots, x_{middle}\}
4. B = \{x_{middle+1}, x_{middle+2}, \ldots, x_n\}
5. As = MERGESORT(A,middle)  T(n/2)
6. Bs = MERGESORT(B,n-middle) T(n/2)
7. RETURN MERGE(As,Bs)  \{0(n)\}
```

Running time:

$T(n) \leq 2T(\frac{n}{2}) + cn \quad n>1$

$T(n) \leq c \quad n=1$
A recurrence

Running time of MergeSort:

How to bound \( T(n) \)?

\[
T(n) \leq 2 T\left(\frac{n}{2}\right) + cn \quad n > 1
\]

\[
T(n) \leq c \quad n = 1
\]

\( (*) \)

\[
T(n) \leq 2 \left[ 2 T\left(\frac{n}{4}\right) + c \frac{n}{2} \right] + cn = 4 T\left(\frac{n}{4}\right) + cn + cn = 4 T\left(\frac{n}{4}\right) + 2cn
\]

\( (*) \)

\[
\leq 4 \left[ 2 T\left(\frac{n}{8}\right) + c \frac{n}{4} \right] + 2cn = 8 T\left(\frac{n}{8}\right) + cn + 2cn = 8 T\left(\frac{n}{8}\right) + 3cn
\]

\( (*) \)

\[
\leq 8 \left[ 2 T\left(\frac{n}{16}\right) + c \cdot \frac{n}{8} \right] + 3cn = 16 T\left(\frac{n}{16}\right) + 4cn
\]

Apply \( (*) \) \( k \) times:

\[
\leq 2^k T\left(\frac{n}{2^k}\right) + kcn
\]

we stop applying \( (*) \) when \( \frac{n}{2^k} = 1 \)

i.e. \( k = \log n \)

then:

\[
T(n) \leq 2^k T\left(\frac{n}{2^k}\right) + kcn = n T(1) + \log n \cdot c \cdot n
\]

\( (k = \log n) \quad = cn + cn \log n = O(n \log n) \) \( \checkmark \)
A recurrence

Running time of MergeSort:

How to bound $T(n)$?

$\rightarrow$ “substitution / induction”

proof our guess:

BASE CASE: $n=1$ : know $T(n) \leq c$ (form (a))

IND. CASE: $n>1$ inductive hypothesis (IH):

Statement holds for $\frac{n}{2}$

i.e.: $T\left(\frac{n}{2}\right) \leq d \frac{n}{2} \cdot \log \frac{n}{2} + d \frac{n}{2}$

know: $T(n) \leq 2T\left(\frac{n}{2}\right) + cn \leq 2 \left[ d \frac{n}{2} \cdot \log \frac{n}{2} + d \frac{n}{2} \right] + cn = dn \left(\log n - 1\right) + dn + cn$

want to show: $T(n) \leq dn \log n + dn$

want to prove: $T(n) \leq dn \log n + dn$

want to show: $T(n) \leq dn \log n + dn$

want to show: $T(n) \leq dn \log n + dn$

want to show: $T(n) \leq dn \log n + dn$

want to show: $T(n) \leq dn \log n + dn$

want to show: $T(n) \leq dn \log n + dn$

want to show: $T(n) \leq dn \log n + dn$

want to show: $T(n) \leq dn \log n + dn$

want to show: $T(n) \leq dn \log n + dn$

want to show: $T(n) \leq dn \log n + dn$
More on sorting

Other $O(n \log n)$ sorts?

Can do better than $O(n \log n)$?
A lower-bound on sorting: $\Omega(n \log n)$

Every comparison-based sort needs at least $\Omega(n \log n)$ comparisons, thus its running time is $\Omega(n \log n)$. 
Counting inversions

Input: A permutation $a_1, a_2, \ldots, a_n$ of $1, 2, \ldots, n$

Output: # inversions, i.e. $(i, j)$ pairs where $i < j$ but $a_i > a_j$
Divide-and-conquer algorithms

**Closest Pair of Points**

Input: n points in the plane \((x_1, y_1), \ldots, (x_n, y_n)\)

Output: distance of the closest pair of points, i.e.,

\[
\min_{i,j \ (i \neq j)} \text{distance of } (x_i, y_i) \text{ and } (x_j, y_j)
\]
Master Theorem

Let $a \geq 1$ and $b > 1$ be constants, $f(n)$ be a function and for positive integers we have a recurrence for $T$ of the form

$$T(n) = a \cdot T(n/b) + f(n),$$

where $n/b$ is rounded either way.

Then,

- If $f(n) = O(n^{\log a / \log b} - e)$ for some constant $e > 0$, then
  $$T(n) = \Theta(n^{\log a / \log b}).$$

- If $f(n) = \Theta(n^{\log a / \log b})$, then
  $$T(n) = \Theta(n^{\log a / \log b} \log n).$$

- If $f(n) = \Omega(n^{\log a / \log b} + e)$ for some constant $e > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ (and all sufficiently large $n$), then
  $$T(n) = \Theta(f(n)).$$

For merge sort:

- $a = 2, b = 2, f(n) = cn, \log a / \log b = 1$
Master Theorem

Let $a \geq 1$ and $b > 1$ be constants, $f(n)$ be a function and for positive integers we have a recurrence for $T$ of the form

$$T(n) = a \cdot T(n/b) + f(n),$$

where $n/b$ is rounded either way.

Then,

- If $f(n) = O(n^{\log a/\log b - \epsilon})$ for some constant $\epsilon > 0$, then
  $$T(n) = \Theta(n^{\log a/\log b}).$$

- If $f(n) = \Omega(n^{\log a/\log b})$, then
  $$T(n) = \Theta(n^{\log a/\log b} \log n).$$

- If $f(n) = \Omega(n^{\log a/\log b + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ (and all sufficiently large $n$), then
  $$T(n) = \Theta(n^2).$$

Another example:

$$T(n) = 2T\left(\frac{n}{2}\right) + n^2$$

Then $a = 2$, $b = 2$, $f(n) = n^2$, $\log a/\log b = 1$.
Master Theorem

Let $a \geq 1$ and $b > 1$ be constants, $f(n)$ be a function and for positive integers we have a recurrence for $T$ of the form

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n),$$

where $n/b$ is rounded either way.

Then,

- If $f(n) = O(n^{\log_a b - e})$ for some constant $e > 0$, then $T(n) = \Theta(n^{\log_a b})$.
- If $f(n) = \Theta(n^{\log_a b})$, then $T(n) = \Theta(n^{\log_a b \log n})$.
- If $f(n) = \Omega(n^{\log_a b + e})$ for some constant $e > 0$, and if $a f(n/b) \leq c f(n)$ for some constant $c < 1$ (and all sufficiently large $n$), then $T(n) = \Theta(f(n))$. 

Another example:

- $T(n) = 2T\left(\frac{n}{2}\right) + c$
  - $a = 2$
  - $b = 2$
  - $\log_a \log_b = 1$
  - $f(n) = c$