We can design algorithms a few ways:
We can design algorithms a few ways:

- incremental – sort up to $j - 1$, then sort up to $j$
We can design algorithms a few ways:

- incremental – sort up to \( j - 1 \), then sort up to \( j \)
- divide and conquer
Divide and conquer

- divide: split problem into subproblems of the same type
- conquer: solve the subproblems recursively
- combine: combine the results into a solution for the original problem
Given an unsorted array, if we could somehow separate it into two sorted arrays, we could then merge these two to get a final sorted array.
Given an unsorted array, if we could somehow separate it into two sorted arrays, we could then merge these two to get a final sorted array.

- Break it into two arrays
Given an unsorted array, if we could somehow separate it into two sorted arrays, we could then merge these two to get a final sorted array.

- Break it into two arrays
- An array of size 1 is already sorted
Suppose we have the following unsorted array:

| 45 | -6 | 73 | 28 | 5  | -1 | 23 | 42 |
Merge sort

Repeatedly divide

45 -6 73 28 5 -1 23 42

45 -6 73 28

45 -6 73 28

5 -1 23 42
Merge sort

Repeatedly divide

45 -6 73 28 5 -1 23 42

45 -6 73 28

5 -1 23 42

45 -6 73 28

5 -1 23 42
Merge sort

Repeatedly divide

45 -6 73 28 5 -1 23 42

45 -6 73 28

45 -6 73 28

45 -6 73 28

45 -6 73 28

45 -6 73 28

45 -6 73 28
Now merge:
Now merge:

```
-6  45
 
45  -6  73  28  5  -1  23  42
```
Now merge:
Now merge:
Merge sort

Now merge:

-6 45
28 73
45 -6 73 28 5 -1 23 42
Merge sort

Now merge:

-6 28 45 73

-6 45

28 73

-1 5

23 42

45

-6

73

28

5

-1

23

42
Merge sort

Now merge:

-6 28 45 73
-6 45
45 -6
45

-1 5 23 42
-1 5
5 -1
23 42
23
Merge sort

Now merge:

-6 -1 5 23 28 42 45 73

-6 28 45 73

-6 45

45 -6 73 28

5 -1 23 42

23 42

-1 5

-1 5 23 42
What is the running time of merge sort?

- Divide $n$ elements into two sets of $n/2$ elements and solve
- Merge the results
How long does it take to merge two sorted $n/2$ lists?

\[
\begin{array}{cccc}
1 & 5 & 10 & 20 \\
3 & 4 & 7 & 42 \\
1 & 3 & 4 & 5 & 7 & 10 & 20 & 42 \\
\end{array}
\]
Merge sort

What is the running time of merge sort?

\[ T(n) = 2T(n/2) + n \]
What is the running time of merge sort?

\[ T(n) = 2T(n/2) + n = 2(2T(n/4) + n/2) + n \]
What is the running time of merge sort?

\[ T(n) = 2T\left(\frac{n}{2}\right) + n = 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n \]
\[ = 4T\left(\frac{n}{4}\right) + 2n \]
Merge sort

What is the running time of merge sort?

\[ T(n) = 2T(n/2) + n = 2(2T(n/4) + n/2) + n \]
\[ = 4T(n/4) + 2n = 4(2T(n/8) + n/4) + 2n \]
What is the running time of merge sort?

\[ T(n) = 2 T(n/2) + n = 2(2 T(n/4) + n/2) + n \]
\[ = 4 T(n/4) + 2n = 4(2 T(n/8) + n/4) + 2n \]
\[ = 8 T(n/8) + 3n \]
Merge sort

What is the running time of merge sort?

\[ T(n) = 2T(n/2) + n = 2(2T(n/4) + n/2) + n \]
\[ = 4T(n/4) + 2n = 4(2T(n/8) + n/4) + 2n \]
\[ = 8T(n/8) + 3n \]
\[ = \ldots \]
What is the running time of merge sort?

\[ T(n) = 2T(n/2) + n = 2(2T(n/4) + n/2) + n \]
\[ = 4T(n/4) + 2n = 4(2T(n/8) + n/4) + 2n \]
\[ = 8T(n/8) + 3n \]
\[ = \ldots \]
\[ = 2^k T(n/2^k) + kn \]
What is the running time of merge sort?

\[ T(n) = 2^k T(n/2^k) + kn \]
What is the running time of merge sort?

\[ T(n) = 2^k T(n/2^k) + kn \]

Let \( n = 2^k \):
What is the running time of merge sort?

$$T(n) = 2^k T(n/2^k) + kn$$

Let $n = 2^k$:

$$T(n) = nT(n/n) + kn$$
What is the running time of merge sort?

\[ T(n) = 2^k T(n/2^k) + kn \]

Let \( n = 2^k \):

\[ T(n) = nT(n/n) + kn \]
\[ = nT(1) + kn \]
What is the running time of merge sort?

\[ T(n) = 2^k T(n/2^k) + kn \]

Let \( n = 2^k \):

\[ T(n) = nT(n/n) + kn = nT(1) + kn = n \cdot 1 + kn \]
What is the running time of merge sort?

\[ T(n) = 2^k T(n/2^k) + kn \]

Let \( n = 2^k \):

\[ T(n) = nT(n/n) + kn \]
\[ = nT(1) + kn \]
\[ = n \cdot 1 + kn \]
\[ = n + kn \]
What is the running time of merge sort?

\[ T(n) = n + kn \]
What is the running time of merge sort?

\[ T(n) = n + kn \]

To get rid of \( k \), observe that \( n = 2^k \) implies \( k = \log n \):
What is the running time of merge sort?

\[ T(n) = n + kn \]

To get rid of \( k \), observe that \( n = 2^k \) implies \( k = \log n \):

\[ T(n) = n + (\log n)n \]
What is the running time of merge sort?

\[ T(n) = n + kn \]

To get rid of \( k \), observe that \( n = 2^k \) implies \( k = \log n \):

\[
T(n) = n + (\log n)n \\
= O(n) + O(n \log n)
\]
What is the running time of merge sort?

\[ T(n) = n + kn \]

To get rid of \( k \), observe that \( n = 2^k \) implies \( k = \log n \):

\begin{align*}
T(n) &= n + (\log n)n \\
&= O(n) + O(n \log n) \\
&= O(n \log n)
\end{align*}
Suppose you invent an AI that correctly predicts the stock market for the next two weeks.

And suppose you want to buy a stock on one day and sell another day.

What are the best days to buy and sell to maximize your profit?
Buying stocks

Stock prices:

<table>
<thead>
<tr>
<th>Day</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Price</td>
<td>100</td>
<td>113</td>
<td>110</td>
<td>85</td>
<td>105</td>
<td>102</td>
<td>86</td>
<td>63</td>
<td>81</td>
<td>101</td>
<td>94</td>
<td>106</td>
<td>101</td>
<td>79</td>
<td>94</td>
<td>90</td>
<td>97</td>
</tr>
<tr>
<td>Change</td>
<td>13</td>
<td>-3</td>
<td>-25</td>
<td>20</td>
<td>-3</td>
<td>-16</td>
<td>-23</td>
<td>18</td>
<td>20</td>
<td>-7</td>
<td>12</td>
<td>-5</td>
<td>-22</td>
<td>15</td>
<td>-4</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>
Buying stocks

First attempt:

- Find the date of the stock’s lowest price, buy then
- Find the highest later price, sell then
But this may not give an optimal result:
Buying stocks

Another case where we don’t get the optimal result:
We could solve this via brute force.

- for each day we could buy:
  - for each (later) day we could sell:
    - calculate the profit
    - keep track of the max
We could solve this via brute force.

```
max = 0
best_i = 0
best_j = 0
for i from 0 to n:
    for j from i+1 to n:
        if a[j] - a[i] > max:
            max = a[j] - a[i]
            best_i = i
            best_j = j
```
Buying stocks

How long would the brute force method take?

- $n - 1$ sell dates if we buy on day 1
- $n - 2$ sell dates if we buy on day 2
- ...
- 2 sell dates if we buy on day $n - 2$
- 1 sell date if we buy on day $n - 1$

For each pair (buyDay, sellDay), we do a constant amount of work ($\Theta(1)$).
How long would the brute force method take?

- There are \( \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \) pairs.
- For each pair (buyDay, sellDay), we do a constant amount of work (\( \Theta(1) \)).
- \( O(n^2) \) total work.
First let’s consider just the price changes each day:

<table>
<thead>
<tr>
<th>Day</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Price</td>
<td>100</td>
<td>113</td>
<td>110</td>
<td>85</td>
<td>105</td>
<td>102</td>
<td>86</td>
<td>63</td>
<td>81</td>
<td>101</td>
<td>94</td>
<td>106</td>
<td>101</td>
<td>79</td>
<td>94</td>
<td>90</td>
<td>97</td>
</tr>
<tr>
<td>Change</td>
<td>13</td>
<td>-3</td>
<td>-25</td>
<td>20</td>
<td>-3</td>
<td>-16</td>
<td>-23</td>
<td>18</td>
<td>20</td>
<td>-7</td>
<td>12</td>
<td>-5</td>
<td>-22</td>
<td>15</td>
<td>-4</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>
Then we’re looking for a subarray with the largest sum:
Maximum subarray problem

If we split the array in two equal parts, where could the maximum subarray lie?

- Entirely in the first half
- Entirely in the second half
- Partly in each, crossing the midpoint

crosses the midpoint

entirely in $A[low..mid]$ entirely in $A[mid + 1..high]$
Maximum subarray problem

If it crosses the midpoint, there must be a part in the left and a part in the right.
Maximum subarray problem

\[ \text{left-sum} = -\infty \]
\[ \text{sum} = 0 \]
\[ \text{for } i = \text{mid} \text{ downto } \text{low} \]
\[ \quad \text{sum} = \text{sum} + A[i] \]
\[ \quad \text{if } \text{sum} > \text{left-sum} \]
\[ \quad \quad \text{left-sum} = \text{sum} \]
\[ \quad \quad \text{max-left} = i \]
\[ \text{right-sum} = -\infty \]
\[ \text{sum} = 0 \]
\[ \text{for } j = \text{mid} + 1 \text{ to } \text{high} \]
\[ \quad \text{sum} = \text{sum} + A[j] \]
\[ \quad \text{if } \text{sum} > \text{right-sum} \]
\[ \quad \quad \text{right-sum} = \text{sum} \]
\[ \quad \quad \text{max-right} = j \]
\[ \text{return } (\text{max-left}, \text{max-right}, \text{left-sum} + \text{right-sum}) \]
if high == low
    return (low, high, A[low])
else mid = \lfloor (low + high)/2 \rfloor
    (left-low, left-high, left-sum) =
        FIND-MAXIMUM-SUBARRAY (A, low, mid)
    (right-low, right-high, right-sum) =
        FIND-MAXIMUM-SUBARRAY (A, mid + 1, high)
    (cross-low, cross-high, cross-sum) =
        FIND-MAX-CROSSING-SUBARRAY (A, low, mid, high)
if left-sum ≥ right-sum and left-sum ≥ cross-sum
    return (left-low, left-high, left-sum)
elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
    return (right-low, right-high, right-sum)
else return (cross-low, cross-high, cross-sum)
How long does this take?

- base case: $T(1) = O(1)$
- recursion: 2 subproblems of size $n/2$
- finding the max crossing subarray: $O(n)$
Maximum subarray problem

How long does this take?

\[
T(n) = \begin{cases} 
O(1) & \text{if } n = 1 \\
2T(n/2) + O(n) & \text{if } n > 1
\end{cases}
\]
Maximum subarray problem

How long does this take?

\[
T(n) = \begin{cases} 
O(1) & \text{if } n = 1 \\
2T(n/2) + O(n) & \text{if } n > 1 
\end{cases}
\]

Same as merge sort! \( O(n \log n) \)
Solving recurrences

- Substitution method
- Recursion tree method
- Master method
Master theorem

Divide and conquer solves a problem of size $n$ by:

- splitting it into $a$ subproblems of size $n/b$
- combining the answers in $O(n^d)$ time

where $a, b, d > 0$
Master theorem

If \( T(n) = aT(n/b) + O(n^d) \) and \( a > 0, b > 1, d \geq 0 \), then

\[
T(n) = \begin{cases} 
O(n^d) & \text{if } d > \log_b a \\
O(n^d \log n) & \text{if } d = \log_b a \\
O(n^{\log_b a}) & \text{if } d < \log_b a 
\end{cases}
\]
Master theorem

Size $n$

Branching factor $a$

Size $n/b$

Size $n/b^2$

Depth $\log_b n$

Size 1

Width $a^{\log_b n} = n^{\log_b a}$
The work done at each level of the tree is

$$a^k \cdot O\left(\frac{n}{b^k}\right)^d = O(n^d) \cdot \left(\frac{a}{b^d}\right)^k$$
Master theorem

\[ O(n^d) \cdot \left( \frac{a}{b^d} \right)^k \]

Taking \( k \) from 0 to \( \log n \) yields a geometric series with ratio \( \frac{a}{b^d} \):

\[ O(n^d) + O(n^d) \cdot \left( \frac{a}{b^d} \right) + O(n^d) \cdot \left( \frac{a}{b^d} \right)^2 + \cdots + O(n^d) \left( \frac{a}{b^d} \right)^{\log_b n} \]
Suppose $c > 0$ and $g(n) = 1 + c + c^2 + \cdots + c^n$.

Then:

- if $c < 1$, $g(n) = \Theta(?)$
- if $c = 1$, $g(n) = \Theta(?)$
- if $c > 1$, $g(n) = \Theta(?)$
Asymptotics of a geometric series

Suppose $c > 0$ and $g(n) = 1 + c + c^2 + \cdots + c^n$.

Then:

- if $c < 1$, $g(n) = \Theta(1)$
- if $c = 1$, $g(n) = \Theta(n)$
- if $c > 1$, $g(n) = \Theta(c^n)$
Master theorem

\[ O(n^d) \cdot \left( \frac{a}{b^d} \right)^k \]

Taking \( k \) from 0 to \( \log n \) yields a geometric series with ratio \( a/b^d \):

- ratio < 1: use first term, \( O(n^d) \)
- ratio = 1: all \( O(\log n) \) terms are \( O(n^d) \), so we have \( O(n^d \log n) \)
- ratio > 1: use last term, \( O(n^{\log_b a}) \)
We can convert logarithm bases by multiplying by a constant factor:

\[ \log_a n = \frac{\log_b n}{\log_b a} \]

so

\[ \log_b n = (\log_a n)(\log_b a) \]
Master theorem

\[ n^d \left( \frac{a}{b^d} \right)^{\log_b n} = n^d \left( \frac{a^{\log_b n}}{(b^{\log_b n})^d} \right) = a^{\log_b n} = a^{(\log_a n)(\log_b a)} = n^{\log_b a} \]
And these three cases correspond to the three cases of the master theorem.

If $T(n) = aT(n/b) + O(n^d)$ and $a > 0$, $b > 1$, $d \geq 0$, then

\[
T(n) = \begin{cases} 
  O(n^d) & \text{if } d > \log_b a \\
  O(n^d \log n) & \text{if } d = \log_b a \\
  O(n^{\log_b a}) & \text{if } d < \log_b a 
\end{cases}
\]
Given a sorted array, use binary search to find an element $k$:

- **Divide:** search for $k$ in either the left or right half
- **Combine:** return the result
Binary search

\[ T(n) = T(n/2) + O(1) \]
Binary search

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

Master theorem: \[ T(n) = aT(n/b) + O(n^d) \]

- \( a = 1 \)
- \( b = 2 \)
- \( d = 0 \)
Binary search

If $T(n) = aT(n/b) + O(n^d)$ and $a > 0$, $b > 1$, $d \geq 0$, then

$$T(n) = \begin{cases} 
O(n^d) & \text{if } d > \log_b a \\
O(n^d \log n) & \text{if } d = \log_b a \\
O(n^{\log_b a}) & \text{if } d < \log_b a 
\end{cases}$$

- $a = 1$
- $b = 2$
- $d = 0$

$$\log_b a = \log_2 1 = 0 = d$$

So we use the second case.
If $T(n) = aT(n/b) + O(n^d)$ and $a > 0$, $b > 1$, $d \geq 0$, then

$$T(n) = \begin{cases} 
O(n^d) & \text{if } d > \log_b a \\
O(n^d \log n) & \text{if } d = \log_b a \\
O(n^{\log_b a}) & \text{if } d < \log_b a
\end{cases}$$

- $a = 1$
- $b = 2$
- $d = 0$

$O(n^d \log n) = O(n^0 \log n) = O(\log n)$
Recall the recurrence for merge sort:

\[ T(n) = 2T(n/2) + O(n) \]

To match \( T(n) = aT(n/b) + O(n^d) \):

- \( a = 2 \)
- \( b = 2 \)
- \( d = 1 \)
If $T(n) = aT(n/b) + O(n^d)$ and $a > 0$, $b > 1$, $d \geq 0$, then

$$T(n) = \begin{cases} 
O(n^d) & \text{if } d > \log_b a \\
O(n^d \log n) & \text{if } d = \log_b a \\
O(n^{\log_b a}) & \text{if } d < \log_b a 
\end{cases}$$

- $a = 2$
- $b = 2$
- $d = 1$

$$\log_b a = \log_2 2 = 1 = d$$

Second case again!
If \( T(n) = aT(n/b) + O(n^d) \) and \( a > 0, b > 1, d \geq 0 \), then

\[
T(n) = \begin{cases} 
O(n^d) & \text{if } d > \log_b a \\
O(n^d \log n) & \text{if } d = \log_b a \\
O(n^{\log_b a}) & \text{if } d < \log_b a 
\end{cases}
\]

- \( a = 2 \)
- \( b = 2 \)
- \( d = 1 \)

\[
O(n^d \log n) = O(n^1 \log n) = O(n \log n)
\]
Recall:

$$A \cdot B = C$$

then $c_{ij}$ is the dot product of row $i$ of $A$ and column $j$ of $B$. 
Matrix multiplication

- How long does the naive multiplication method take?
- Can we view this as a divide and conquer algorithm?
Matrix multiplication

Suppose we decompose the matrices into blocks:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$
Matrix multiplication

Then we can define multiplication as such:

\[
\begin{pmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{pmatrix}
= \begin{pmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{pmatrix}
\cdot \begin{pmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{pmatrix}
\]
Matrix multiplication

So we have these subproblems to compute:

\[
\begin{align*}
C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21} \\
C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\
C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21} \\
C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22}
\end{align*}
\]
Matrix multiplication

\[ n = A.\text{rows} \]
let \( C \) be a new \( n \times n \) matrix

\textbf{if} \( n == 1 \)
\[ c_{11} = a_{11} \cdot b_{11} \]
\textbf{else} partition \( A, B, \) and \( C \) as in equations (4.9)
\[ C_{11} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{11}, B_{11}) + \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{12}, B_{21}) \]
\[ C_{12} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{11}, B_{12}) + \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{12}, B_{22}) \]
\[ C_{21} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{21}, B_{11}) + \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{22}, B_{21}) \]
\[ C_{22} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{21}, B_{12}) + \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{22}, B_{22}) \]

return \( C \)
Matrix multiplication

- Divide into 8 blocks of size $n/2 \times n/2$
- Recursively multiply
- Add (some of) the resulting matrices
Matrix multiplication

Then the time required has the form:

\[ T(n) = 8T(n/2) + O(n^2) \]
Matrix multiplication

\[ T(n) = 8T(n/2) + O(n^2) \]

Master theorem: \( T(n) = aT(n/b) + O(n^d) \)

- \( a = 8 \)
- \( b = 2 \)
- \( d = 2 \)
Matrix multiplication

If $T(n) = aT(n/b) + O(n^d)$ and $a > 0$, $b > 1$, $d \geq 0$, then

$$T(n) = \begin{cases} 
O(n^d) & \text{if } d > \log_b a \\
O(n^d \log n) & \text{if } d = \log_b a \\
O(n^{\log_b a}) & \text{if } d < \log_b a 
\end{cases}$$

- $a = 8$
- $b = 2$
- $d = 2$

\[\log_b a = \log_2 8 = 3 > 2 = d\]

Use case 3.
If \( T(n) = aT(n/b) + O(n^d) \) and \( a > 0, b > 1, d \geq 0 \), then

\[
T(n) = \begin{cases} 
O(n^d) & \text{if } d > \log_b a \\
O(n^d \log n) & \text{if } d = \log_b a \\
O(n^{\log_b a}) & \text{if } d < \log_b a
\end{cases}
\]

- \( a = 8 \)
- \( b = 2 \)
- \( d = 2 \)

\[ O(n^{\log_b a}) = O(n^3) \]