CS 344: Design & Analysis of Algorithms

Lecture 5

Sep 17, 2019
$n = 2^k$, which gave us the recurrence

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

\[ = O(n \log n) \]
Merge sort

\[ n = 2^k + 2c \]
Then on the last full level, there are $2^k - c$ leaf nodes and $c$ nodes with size 2.

\[ T(n) = (2^k - c) \cdot T(1) + c \cdot T(2) + kn \]
Merge sort

\[ T(n) = (2^k - c) \cdot T(1) + c \cdot T(2) + kn \]
\[ = (2^k - c) \cdot 1 + c \cdot 4 + kn \]
\[ = 2^k + 3c + kn \]
Since $n = 2^k + c$, we have $k = \lg(n - c)$

$$T(n) = 2^k + 3c + kn = 2^{\lg(n-c)} + 3c + \lg(n - c) \cdot n$$

$$= n + 2c + n \lg(n - c)$$

$$= O(n \lg n)$$
### Maximum subarray problem

The maximum subarray problem is a classic algorithmic problem in computer science. It involves finding the contiguous subarray within a one-dimensional array of numbers which has the largest sum. Here is an example array:

\[
\]

The maximum subarray in this case is `[20, -7, 12, -5, -22, 15, -4, 7]` with a sum of `23`.
Note that any subarray must end at some index $i$.

Let's consider the problem inductively:

- Suppose we know the maximum subarray ending at index $i$
- Can we figure out the maximum subarray ending at $i + 1$?
Maximum subarray problem

This is known as Kadane’s algorithm.

```python
def maxSubarray(numbers):
    bestSum = 0
    currSum = 0
    for x in numbers:
        currSum = max(0, currSum + x)
        bestSum = max(bestSum, currSum)
    return bestSum
```
Maximum subarray problem

Consider the array 4 -6 7 2 5 -1 3 2:

<table>
<thead>
<tr>
<th>i</th>
<th>A[i]</th>
<th>current sum</th>
<th>best sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>-6</td>
<td>-6</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>-1</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>18</td>
<td>18</td>
</tr>
</tbody>
</table>
### Maximum subarray problem

Consider the array $4\ -6\ 7\ 2\ 5\ -1\ 3\ 2$:

<table>
<thead>
<tr>
<th>i</th>
<th>$A[i]$</th>
<th>current sum</th>
<th>best sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>$\max(0, 4) = 4$</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>-6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Consider the array 4 -6 7 2 5 -1 3 2:

<table>
<thead>
<tr>
<th>i</th>
<th>A[i]</th>
<th>current sum</th>
<th>best sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>max(0, 4) = 4</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>-6</td>
<td>max(0, -2) = 0</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Maximum subarray problem

Consider the array $4\ -6\ 7\ 2\ 5\ -1\ 3\ 2$:

<table>
<thead>
<tr>
<th></th>
<th>$A[i]$</th>
<th>current sum</th>
<th>best sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>$\max(0, 4) = 4$</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>-6</td>
<td>$\max(0, -2) = 0$</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>$\max(0, 7) = 7$</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Maximum subarray problem

Consider the array 4 -6 7 2 5 -1 3 2:

<table>
<thead>
<tr>
<th>i</th>
<th>A[i]</th>
<th>current sum</th>
<th>best sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>max(0, 4) = 4</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>-6</td>
<td>max(0, -2) = 0</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>max(0, 7) = 7</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>max(0, 9) = 9</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Maximum subarray problem**

Consider the array $4 \ -6 \ 7 \ 2 \ 5 \ -1 \ 3 \ 2$:

<table>
<thead>
<tr>
<th>$i$</th>
<th>$A[i]$</th>
<th>current sum</th>
<th>best sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>max(0, 4) = 4</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>-6</td>
<td>max(0, -2) = 0</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>max(0, 7) = 7</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>max(0, 9) = 9</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>max(0, 14) = 14</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Maximum subarray problem

Consider the array 4 -6 7 2 5 -1 3 2:

<table>
<thead>
<tr>
<th>i</th>
<th>A[i]</th>
<th>current sum</th>
<th>best sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>max(0, 4) = 4</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>-6</td>
<td>max(0, -2) = 0</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>max(0, 7) = 7</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>max(0, 9) = 9</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>max(0, 14) = 14</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>-1</td>
<td>max(0, 13) = 13</td>
<td>14</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Consider the array 4 -6 7 2 5 -1 3 2:

<table>
<thead>
<tr>
<th>i</th>
<th>A[i]</th>
<th>current sum</th>
<th>best sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>max(0, 4) = 4</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>-6</td>
<td>max(0, -2) = 0</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>max(0, 7) = 7</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>max(0, 9) = 9</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>max(0, 14) = 14</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>-1</td>
<td>max(0, 13) = 13</td>
<td>14</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>max(0, 16) = 16</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Maximum subarray problem

Consider the array 4 -6 7 2 5 -1 3 2:

<table>
<thead>
<tr>
<th>i</th>
<th>A[i]</th>
<th>current sum</th>
<th>best sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>max(0, 4) = 4</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>-6</td>
<td>max(0, -2) = 0</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>max(0, 7) = 7</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>max(0, 9) = 9</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>max(0, 14) = 14</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>-1</td>
<td>max(0, 13) = 13</td>
<td>14</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>max(0, 16) = 16</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>max(0, 18) = 18</td>
<td>18</td>
</tr>
</tbody>
</table>
Kadane’s algorithm:

```python
def maxSubarray(numbers):
    bestSum = 0
    currSum = 0
    for x in numbers:
        currSum = max(0, currSum + x)
        bestSum = max(bestSum, currSum)
    return bestSum
```

What is the running time?
Another method to solve recurrences is the “substitution method”:

- Guess the correct form
- Prove inductively
Substitution method

Given the following recurrence:

\[ T(n) = \begin{cases} 
1 & \text{if } n = 1 \\
2T(\lfloor n/2 \rfloor) + n & \text{otherwise} 
\end{cases} \]
Substitution method

Given the following recurrence:

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

Since it's similar to \(2T(n/2) + n\), we could guess that it's also \(O(n \lg n)\).
Substitution method

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

To prove it's \(O(n \lg n)\), we must show \(T(n) \leq cn \lg n\) for some \(c > 0\), for sufficiently large \(n\).
Substitution method

Inductive step: assume this is true for all $m < n$

Then it’s true for $m = \lfloor n/2 \rfloor$,

so we have $T(\lfloor n/2 \rfloor) \leq c \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)$

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$
Inductive step: assume this is true for all $m < n$

Then it’s true for $m = \lfloor n/2 \rfloor$,
so we have $T(\lfloor n/2 \rfloor) \leq c \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)$

\[
T(n) = 2T(\lfloor n/2 \rfloor) + n \\
\leq 2(c \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + n
\]
Inductive step: assume this is true for all $m < n$

Then it’s true for $m = \lfloor n/2 \rfloor$,

so we have $T(\lfloor n/2 \rfloor) \leq c \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)$

$$\begin{align*}
T(n) &= 2T(\lfloor n/2 \rfloor) + n \\
&\leq 2(c \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + n \\
&\leq cn\lg(n/2) + n
\end{align*}$$
Substitution method

Inductive step: assume this is true for all \( m < n \)
Then it’s true for \( m = \lfloor n/2 \rfloor \),
so we have \( T(\lfloor n/2 \rfloor) \leq c \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor) \)

\[
T(n) = 2T(\lfloor n/2 \rfloor) + n \\
\leq 2(c \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + n \\
\leq cn \lg(n/2) + n \\
= cn \lg n - cn \lg 2 + n
\]
Substitution method

Inductive step: assume this is true for all $m < n$

Then it’s true for $m = \lfloor n/2 \rfloor$,

so we have $T(\lfloor n/2 \rfloor) \leq c \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)$

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$

$$\leq 2(c \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + n$$

$$\leq cn \lg(n/2) + n$$

$$= cn \lg n - cn \lg 2 + n$$

$$= cn \lg n - cn + n$$
Inductive step: assume this is true for all $m < n$

Then it’s true for $m = \lfloor n/2 \rfloor$,

so we have $T(\lfloor n/2 \rfloor) \leq c \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)$

\[
T(n) = 2T(\lfloor n/2 \rfloor) + n \\
\leq 2(c \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + n \\
\leq cn \lg(n/2) + n \\
= cn \lg n - cn \lg 2 + n \\
= cn \lg n - cn + n \\
= cn \lg n - (c - 1)n
\]
Substitution method

Inductive step: assume this is true for all $m < n$

Then it’s true for $m = \lfloor n/2 \rfloor$,

so we have $T(\lfloor n/2 \rfloor) \leq c \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)$

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$

$$\leq 2(c \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + n$$

$$\leq cn \lg(n/2) + n$$

$$= cn \lg n - cn \lg 2 + n$$

$$= cn \lg n - cn + n$$

$$= cn \lg n - (c - 1)n$$

$$\leq cn \lg n$$
Base case: we’re given \( T(1) = 1 \), and must show

\[
T(1) \leq cn \lg n
\]

But this equals \( c1 \lg 1 = 0 \)…
Substitution method

Fortunately, we only need this to hold for $n > n_0$

By the recurrence, $T(2) = 2T(1) + 2 = 2 + 2 = 4$.

$$T(2) \leq c2 \log 2 = 2c$$

for any $c \geq 2$. 
Fortunately, we only need this to hold for $n > n_0$

By the recurrence, $T(2) = 2T(1) + 2 = 2 + 2 = 4$.

$$T(2) \leq c2 \lg 2 = 2c$$

for any $c \geq 2$.

Are we done?
Consider $n = 3$:

$$T(3) = 2T(1) + 3$$

But we haven’t proved the $n = 1$ case!
Substitution method

We can handle this as a second base case:

\[ T(3) = 2T(1) + 3 = 5. \]

Then

\[ T(3) \leq c2 \log 3 \approx 3.17c \]

Again, \( c \geq 2 \) suffices.
Sometimes this is a little tricky...

\[
T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1
\]

This is similar to this recurrence:

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

So we could guess that this is also \(O(n)\).
Attempt 1 with induction:

Assume it’s $T(m) \leq cm$ for all $m < n$. Then

$$T(n) \leq c \lfloor n/2 \rfloor + c \lceil n/2 \rceil + 1$$
Attempt 1 with induction:

Assume it’s $T(m) \leq cm$ for all $m < n$. Then

\[
T(n) \leq c \lfloor n/2 \rfloor + c \lceil n/2 \rceil + 1
\]

\[
= cn + 1
\]

But this is not $\leq cn$ for any $c > 0$
Attempt 2: guess $T(n) \leq cn - d$ instead.

Assume it’s $T(m) \leq cm - d$ for all $m < n$. Then

$$T(n) \leq c \lfloor n/2 \rfloor - d + c \lceil n/2 \rceil - d + 1$$
Substitution method

Attempt 2: guess $T(n) \leq cn - d$ instead.

Assume it’s $T(m) \leq cm - d$ for all $m < n$. Then

$$T(n) \leq c \left\lfloor \frac{n}{2} \right\rfloor - d + c \left\lceil \frac{n}{2} \right\rceil - d + 1$$

$$= cn - 2d + 1$$
Substitution method

Attempt 2: guess $T(n) \leq cn - d$ instead.

Assume it’s $T(m) \leq cm - d$ for all $m < n$. Then

$$T(n) \leq c \lfloor n/2 \rfloor - d + c \lceil n/2 \rceil - d + 1$$

$$= cn - 2d + 1$$

$$\leq cn - d$$
Lower bound for sorting

```
<table>
<thead>
<tr>
<th></th>
<th>a₁ &lt; a₂?</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>a₁ &lt; a₃?</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a₂ &lt; a₃?</td>
<td></td>
</tr>
<tr>
<td>3 2 1</td>
<td>2 3 1</td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>a₂ &lt; a₃?</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a₁ &lt; a₃?</td>
<td></td>
</tr>
<tr>
<td>1 3 2</td>
<td>1 2 3</td>
<td></td>
</tr>
</tbody>
</table>
```
Lower bound for sorting

- Every permutation must be considered
Lower bound for sorting

- Every permutation must be considered
- There are $n!$ permutations
Lower bound for sorting

- Every permutation must be considered
- There are $n!$ permutations
- So tree must have at least $n!$ leaves
• A tree of depth $d$ has $\leq 2^d$ leaves
Lower bound for sorting

- A tree of depth \( d \) has \( \leq 2^d \) leaves
- So we must have a depth of at least \( \log(n!) \)
And $\log(n!) \geq cn \log n$ for some $c > 0$

Note that $n! \geq (n/2)^{n/2}$ since $n! = 1 \cdot 2 \cdots n$ has at least $n/2$ factors larger than $n/2$

$$\log(n!) \geq \log((n/2)^{n/2})$$
And $\log(n!) \geq c n \log n$ for some $c > 0$.

Note that $n! \geq (n/2)^{n/2}$ since $n! = 1 \cdot 2 \cdots n$ has at least $n/2$ factors larger than $n/2$.

$$
\log(n!) \geq \log((n/2)^{n/2})
= (n/2) \log(n/2)
$$
And $\log(n!) \geq cn \log n$ for some $c > 0$

Note that $n! \geq (n/2)^{n/2}$ since $n! = 1 \cdot 2 \cdots n$ has at least $n/2$ factors larger than $n/2$

\[
\log(n!) \geq \log\left((n/2)^{n/2}\right)
= (n/2) \log(n/2)
= (n/2)(\log n - \log 2)
\]
Lower bound for sorting

And \( \log(n!) \geq cn \log n \) for some \( c > 0 \)

Note that \( n! \geq (n/2)^{n/2} \) since \( n! = 1 \cdot 2 \cdots n \) has at least \( n/2 \) factors larger than \( n/2 \)

\[
\log(n!) \geq \log((n/2)^{n/2}) \\
= (n/2) \log(n/2) \\
= (n/2)(\log n - \log 2) \\
= 1/2n \log n - (n/2) \log 2
\]
And $\log(n!) \geq cn \log n$ for some $c > 0$

Note that $n! \geq (n/2)^{n/2}$ since $n! = 1 \cdot 2 \cdots n$ has at least $n/2$ factors larger than $n/2$

$$
\log(n!) \geq \log((n/2)^{n/2})
= (n/2) \log(n/2)
= (n/2)(\log n - \log 2)
= 1/2n \log n - (n/2) \log 2
\geq 1/2n \log n
$$
Thus we must make $\Omega(n \log n)$ comparisons to reach a leaf.
Thus we must make $\Omega(n \log n)$ comparisons to reach a leaf. So any sorting algorithm must take $\Omega(n \log n)$ time.
Lower bound for sorting

Thus we must make $\Omega(n \log n)$ comparisons to reach a leaf.
So any sorting algorithm must take $\Omega(n \log n)$ time.
Then merge sort is optimal, with $\Theta(n \log n)$ time.
Quicksort

- Divide: partition A[p..r] into A[p..q-1] and A[q+1..r] such that the first is all ≤ A[q] and the second is all ≥ A[q]
- Conquer: recursively sort the two subarrays

The value A[q] is called the pivot.
Quicksort

QUICKSORT(A, p, r)
1   if p < r
2       q = PARTITION(A, p, r)
3       QUICKSORT(A, p, q − 1)
4       QUICKSORT(A, q + 1, r)
Quicksort

\textbf{PARTITION}(A, p, r)

1 \hspace{0.5em} x = A[r] \\
2 \hspace{0.5em} i = p - 1 \\
3 \hspace{0.5em} \textbf{for} \ j = p \ \textbf{to} \ r - 1 \\
4 \hspace{1.5em} \textbf{if} \ A[j] \leq x \\
5 \hspace{2em} i = i + 1 \\
6 \hspace{1.5em} \text{exchange } A[i] \text{ with } A[j] \\
7 \hspace{0.5em} \text{exchange } A[i + 1] \text{ with } A[r] \\
8 \hspace{0.5em} \textbf{return} \ i + 1
Quicksort
Suppose we have the following array:

\[
\begin{array}{cccccccc}
2 & 8 & 7 & 1 & 3 & 5 & 6 & 4
\end{array}
\]
Quicksort
Quicksort

\textsc{Partition}(A, p, r)

1 \quad x = A[r]
2 \quad i = p - 1
3 \quad \textbf{for} j = p \textbf{ to } r - 1
4 \quad \quad \textbf{if} A[j] \leq x
5 \quad \quad \quad i = i + 1
6 \quad \text{exchange } A[i] \text{ with } A[j]
7 \quad \text{exchange } A[i + 1] \text{ with } A[r]
8 \quad \textbf{return} \ i + 1
What is the running time?
What is the running time?

- How long does partitioning take?
What is the running time?

- How long does partitioning take?
- What’s the worst case?
Quicksort

What is the running time?

- How long does partitioning take?
- What’s the worst case?
- Best case?
What is the running time (worst case)?

- Partitioning gives a maximally unbalanced set of regions
What is the running time (worst case)?

- Partitioning gives a maximally unbalanced set of regions
- Since we use one element as the pivot, $n - 1$ elements remain unsorted
Quicksort

What is the running time (worst case)?

- Partitioning gives a maximally unbalanced set of regions
- Since we use one element as the pivot, \( n - 1 \) elements remain unsorted

\[
T(n) = T(n - 1) + O(n)
\]
What is the running time (worst case)?

- Partitioning gives a maximally unbalanced set of regions
- Since we use one element as the pivot, $n - 1$ elements remain unsorted

$$T(n) = T(n - 1) + O(n)$$

$$T(n) = O(n^2)$$
What is the running time (best case)?

- Partitioning gives a balanced set of regions
What is the running time (best case)?

- Partitioning gives a balanced set of regions
- Each side has roughly $n/2$ elements
Quicksort

What is the running time (best case)?

- Partitioning gives a balanced set of regions
- Each side has roughly $n/2$ elements

$$T(n) = 2T(n/2) + O(n)$$
What is the running time (best case)?

- Partitioning gives a balanced set of regions
- Each side has roughly $n/2$ elements

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

\[
T(n) = O(n \log n)
\]