This is CS50
searching
input $\rightarrow$ output
algorithm
linear search
For each door from left to right
    If 50 is behind door
        Return true
Return false
For each door from left to right
   If 50 is behind door
       Return true
   Else
       Return false
For each door from left to right
   If 50 is behind door
      Return true
Return false
For i from 0 to n-1
    If 50 is behind doors[i]
        Return true
Return false
binary search
If 50 is behind middle door
    Return true
Else if 50 < middle door
    Search left half
Else if 50 > middle door
    Search right half
If no doors left

If 50 is behind middle door
  Return true
Else if 50 < middle door
  Search left half
Else if 50 > middle door
  Search right half
If no doors left
    Return false
If 50 is behind middle door
    Return true
Else if 50 < middle door
    Search left half
Else if 50 > middle door
    Search right half
If no doors left
   Return false
If 50 is behind doors[middle]
   Return true
Else if 50 < doors[middle]
   Search doors[0] through doors[middle - 1]
Else if 50 > doors[middle]
   Search doors[middle + 1] through doors[n - 1]
running time
The time to solve a problem as a function of the size of the problem is shown in the graph.

- Red line: $O(n)$
- Yellow line: $O(n/2)$
- Green line: $O(\log_2 n)$
The graph shows the time to solve as a function of the size of the problem. The time complexities are:

- $O(n)$
- $O(n/2)$
- $O(\log_2 n)$

The graph illustrates how the time to solve increases with the size of the problem for each complexity class.
The graph shows the time to solve a problem as a function of the size of the problem. The time complexity is given by:

- $O(n)$ for the red line
- $O(n)$ for the yellow line
- $O(\log_2 n)$ for the green line

The x-axis represents the size of the problem, and the y-axis represents the time to solve the problem.
The graph shows the time to solve for different sizes of problems. The red line represents $O(n)$, the yellow line represents $O(n)$, and the green line represents $O(\log n)$. The x-axis represents the size of the problem, and the y-axis represents the time to solve.
The graph illustrates the relationship between the size of the problem and the time to solve it. The red line represents $O(n)$ complexity, indicating linear growth with problem size. The green line represents $O(\log n)$ complexity, showing logarithmic growth. This suggests that as the size of the problem increases, the time to solve it grows much more slowly for problems with logarithmic complexity compared to those with linear complexity.
$O(n^2)$
$O(n \log n)$
$O(n)$
$O(1)$
\(O(n^2)\)

\(O(n \log n)\)

\(O(n)\)  \hspace{1cm} \text{linear search}

\(O(\log n)\)

\(O(1)\)
$O(n^2)$
$O(n \log n)$
$O(n)$  linear search
$O(\log n)$  binary search
$O(1)$
\( \Omega(n^2) \)

\( \Omega(n \log n) \)

\( \Omega(n) \)

\( \Omega(\log n) \)

\( \Omega(1) \)
$\Omega(n^2)$

$\Omega(n \log n)$

$\Omega(n)$

$\Omega(\log n)$

$\Omega(1)$

linear search
$\Omega(n^2)$

$\Omega(n \log n)$

$\Omega(n)$

$\Omega(\log n)$

$\Omega(1)$

linear search, binary search
$\Theta(n^2)$
$\Theta(n \log n)$
$\Theta(n)$
$\Theta(\log n)$
$\Theta(1)$
linear search
data structures
person people[]
string name;
string number;
typedef struct {
    string name;
    string number;
} person;
unsorted → □ → output
unsorted $\rightarrow$ \hspace{1in} $\rightarrow$ sorted
selection sort
For $i$ from 0 to $n-1$
   Find smallest number between $\text{numbers}[i]$ and $\text{numbers}[n-1]$
   Swap smallest number with $\text{numbers}[i]$
$(n - 1)$
(n - 1) + (n - 2)
\((n - 1) + (n - 2) + (n - 3)\)
$(n - 1) + (n - 2) + (n - 3) + ... + 1$
\[(n - 1) + (n - 2) + (n - 3) + \ldots + 1\]

\[n(n - 1)/2\]
$(n - 1) + (n - 2) + (n - 3) + ... + 1$

$n(n - 1)/2$

$(n^2 - n)/2$
\[(n - 1) + (n - 2) + (n - 3) + \ldots + 1\]
\[n(n - 1)/2\]
\[(n^2 - n)/2\]
\[n^2/2 - n/2\]
\[(n - 1) + (n - 2) + (n - 3) + \ldots + 1\]
\[n(n - 1)/2\]
\[(n^2 - n)/2\]
\[n^2/2 - n/2\]
\[O(n^2)\]
$O(n^2)$

$O(n \log n)$

$O(n)$

$O(\log n)$

$O(1)$
$O(n^2)$ selection sort

$O(n \log n)$

$O(n)$

$O(\log n)$

$O(1)$
For i from 0 to n-1
  Find smallest number between numbers[i] and numbers[n-1]
  Swap smallest number with numbers[i]
\( \Omega(n^2) \)
\( \Omega(n \log n) \)
\( \Omega(n) \)
\( \Omega(\log n) \)
\( \Omega(1) \)
$\Omega(n^2)$ selection sort

$\Omega(n \log n)$

$\Omega(n)$

$\Omega(\log n)$

$\Omega(1)$
\( \Theta(n^2) \)
\( \Theta(n \log n) \)
\( \Theta(n) \)
\( \Theta(n) \)
\( \Theta(\log n) \)
\( \Theta(1) \)
\( \Theta(n^2) \) selection sort

\( \Theta(n \log n) \)

\( \Theta(n) \)

\( \Theta(n) \)

\( \Theta(\log n) \)

\( \Theta(1) \)
bubble sort
Repeat n times
   For i from 0 to n-2
      If numbers[i] and numbers[i+1] out of order
         Swap them
Repeat n-1 times
   For i from 0 to n-2
      If numbers[i] and numbers[i+1] out of order
         Swap them
\[(n - 1) \times (n - 1)\]

\[n^2 - 1n - 1n + 1\]
\((n - 1) \times (n - 1)\)

\(n^2 - 1n - 1n + 1\)

\(n^2 - 2n + 1\)
\[(n - 1) \times (n - 1)\]

\[n^2 - 1n - 1n + 1\]

\[n^2 - 2n + 1\]

\[O(n^2)\]
\(O(n^2)\)
\(O(n \log n)\)
\(O(n)\)
\(O(\log n)\)
\(O(1)\)
\(O(n^2)\)  bubble sort
\(O(n \log n)\)
\(O(n)\)
\(O(1)\)
Repeat n-1 times
   For i from 0 to n-2
      If numbers[i] and numbers[i+1] out of order
         Swap them
Repeat n-1 times
   For i from 0 to n-2
      If numbers[i] and numbers[i+1] out of order
         Swap them
   If no swaps
      Quit
\( \Omega(n^2) \)
\( \Omega(n \log n) \)
\( \Omega(n) \)  bubble sort
\( \Omega(\log n) \)
\( \Omega(1) \)
recursion
If no doors left
    Return false
If number behind middle door
    Return true
Else if number < middle door
    Search left half
Else if number > middle door
    Search right half
If no doors left
   Return false
If number behind middle door
   Return true
Else if number < middle door
   Search left half
Else if number > middle door
   Search right half
Pick up phone book
Open to middle of phone book
Look at page
If person is on page
    Call person
Else if person is earlier in book
    Open to middle of left half of book
    Go back to line 3
Else if person is later in book
    Open to middle of right half of book
    Go back to line 3
Else
    Quit
1 Pick up phone book
2 Open to middle of phone book
3 Look at page
4 If person is on page
5    Call person
6 Else if person is earlier in book
7    Open to middle of left half of book
8    Go back to line 3
9 Else if person is later in book
10   Open to middle of right half of book
11   Go back to line 3
12 Else
13 Quit
1  Pick up phone book
2  Open to middle of phone book
3  Look at page
4  If person is on page
5      Call person
6  Else if person is earlier in book
7      Open to middle of left half of book
8      Go back to line 3
9  Else if person is later in book
10     Open to middle of right half of book
11     Go back to line 3
12  Else
13      Quit
1   Pick up phone book
2   Open to middle of phone book
3   Look at page
4   If person is on page
5       Call person
6   Else if person is earlier in book
7       Search left half of book
8
9   Else if person is later in book
10       Search right half of book
11
12   Else
13       Quit
1 Pick up phone book
2 Open to middle of phone book
3 Look at page
4 If person is on page
5   Call person
6 Else if person is earlier in book
7   Search left half of book
8 Else if person is later in book
9   Search right half of book
10 Else
11   Quit
google.com/search?q=recursion
merge sort
Sort left half of numbers
Sort right half of numbers
Merge sorted halves
If only one number
   Quit
Else
   Sort left half of numbers
   Sort right half of numbers
   Merge sorted halves
If only one number
   Quit
Else
   Sort left half of numbers
   Sort right half of numbers
   Merge sorted halves
If only one number
   Quit
Else
   Sort left half of numbers
   Sort right half of numbers
   Merge sorted halves
<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>
$O(n^2)$

$O(n \log n)$ merge sort

$O(n)$

$O(\log n)$

$O(1)$
\[ \Omega(n^2) \]
\[ \Omega(n \log n) \quad \text{merge sort} \]
\[ \Omega(n) \]
\[ \Omega(\log n) \]
\[ \Omega(1) \]
Θ(n^2)

Θ(n \log n)

merge sort

Θ(n)

Θ(\log n)

Θ(1)
time
space
This is CS50