This is CS50
you
I'm hoping you can help me solve a problem

ddb
quack
searching
input → [black square] → output
true
algorithms
running times
The graph shows the relationship between the size of a problem and the time it takes to solve it. The axes are labeled as follows:

- Y-axis: Time to solve
- X-axis: Size of problem

There are three lines on the graph, each representing a different scenario or algorithm:

- Red line: Represents a scenario where the time to solve grows linearly with the size of the problem.
- Yellow line: Represents a scenario where the time to solve grows more rapidly than linearly, possibly indicating a quadratic growth.
- Green line: Represents a scenario where the time to solve grows more slowly, indicating a slower growth rate than the linear or quadratic scenarios.

The graph illustrates how the complexity of solving a problem can increase at different rates depending on the algorithm or approach used.
The time to solve a problem depends on its size. For a problem of size \( n \), the time is \( O(n) \). For a problem of size \( n/2 \), the time is \( O(n/2) \). When the size of the problem is \( \log_2 n \), the time also grows slowly, indicating a more efficient algorithm compared to linear or logarithmic time.
The size of the problem affects the time to solve.

- $O(n)$
- $O(n/2)$
- $O(\log_2 n)$
The graph shows the time to solve as a function of the size of the problem.

- The red line with slope 1 represents $O(n)$.
- The yellow line with slope 1 represents $O(n)$.
- The green line with slope $\log_2 n$ represents $O(\log_2 n)$.

These represent different complexities in terms of time required to solve a problem as the size $n$ increases.

- $O(n)$ means the time increases linearly with the size.
- $O(\log_2 n)$ means the time increases logarithmically with the size.
The time to solve a problem grows with the size of the problem. For a linear problem, the time is $O(n)$. For a quadratic problem, the time is $O(n^2)$. For a logarithmic problem, the time is $O(\log n)$. These are represented graphically with red, yellow, and green lines, respectively.
The graph shows the relationship between the size of a problem and the time it takes to solve it, with two different time complexities:

- $O(n)$, represented by the red line, indicates linear time complexity.
- $O(\log n)$, represented by the green line, indicates logarithmic time complexity.

As the size of the problem increases, the time to solve it grows linearly for $O(n)$, while it grows logarithmically for $O(\log n)$. This implies that for large problem sizes, algorithms with logarithmic time complexity are more efficient than those with linear time complexity.
\( O(n^2) \)
\( O(n \log n) \)
\( O(n) \)
\( O(\log n) \)
\( O(1) \)
$\Omega(n^2)$

$\Omega(n \log n)$

$\Omega(n)$

$\Omega(\log n)$

$\Omega(1)$
linear search
For i from 0 to n-1
    If number behind i'th door
        Return true
    Return false
Return false
\( O(n^2) \)
\( O(n \log n) \)
\( O(n) \)
\( O(\log n) \)
\( O(1) \)
$O(n^2)$

$O(n \log n)$

$O(n)$  linear search

$O(\log n)$

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

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
    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
$O(n^2)$

$O(n \log n)$

$O(n)$  linear search

$O(\log n)$

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

\( O(n \log n) \)

\( O(n) \)        linear search

\( O(\log n) \)   binary search

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

$\Omega(n \log n)$

$\Omega(n)$

$\Omega(\log n)$

$\Omega(1)$  linear search, binary search
int numbers[]
string names[]
data structures
person people[]
string name;
string number;
typedef struct
{
    string name;
    string number;
}
person;
sorting
unsorted $\rightarrow$ output
unsorted $\rightarrow$ sorted
selection sort
For $i$ from 0 to $n-1$
Find smallest item between $i$'th item and last item
Swap smallest item with $i$'th item
\[ n + (n - 1) \]
\[ n + (n - 1) + (n - 2) \]
n + (n – 1) + (n – 2) + ... + 1
\[ n + (n - 1) + (n - 2) + \ldots + 1 \]

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

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

\[ (n^2 + n)/2 \]

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

\(O(n \log n)\)

\(O(n)\)  linear search

\(O(\log n)\)  binary search

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

$O(n \log n)$

$O(n)$ linear search

$O(\log n)$ binary search

$O(1)$
For $i$ from 0 to $n-1$

Find smallest item between $i$'th item and last item
Swap smallest item with $i$'th item
$\Omega(n^2)$

$\Omega(n \log n)$

$\Omega(n)$

$\Omega(\log n)$

$\Omega(1)$  linear search, binary search
Ω(n^2)  selection sort
Ω(n \log n)
Ω(n)
Ω(\log n)
Ω(1)  linear search, binary search
bubble sort
Repeat until sorted
  For i from 0 to n-2
    If i'th and i+1'th elements out of order
      Swap them
Repeat n-1 times
  For i from 0 to n-2
    If i'th and i+1'th elements out of order
      Swap them
\[(n - 1) \times (n - 1)\]
$$(n - 1) \times (n - 1)$$

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

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

\[n^2 - 2n + 1\]
\[(n - 1) \times (n - 1)\]
\[n^2 - 1n - 1n + 1\]
\[n^2 - 2n + 1\]
\[O(n^2)\]
$O(n^2)$ selection sort

$O(n \log n)$

$O(n)$ linear search

$O(\log n)$ binary search

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

$O(n \log n)$

$O(n)$  
linear search

$O(\log n)$  
binary search

$O(1)$
Repeat n-1 times
  For i from 0 to n-2
    If i'th and i+1'th elements out of order
      Swap them
  If no swaps
    Quit
$\Omega(n^2)$ selection sort
$\Omega(n \log n)$
$\Omega(n)$
$\Omega(\log n)$
$\Omega(1)$ linear search, binary search
$\Omega(n^2)$  selection sort

$\Omega(n \log n)$

$\Omega(n)$  bubble sort

$\Omega(\log n)$

$\Omega(1)$  linear search, binary search
recursion
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       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
   Call person
5  Else if person is earlier in book
6    Search left half of book
7  Else if person is later in book
8    Search right half of book
9  Else
10   Quit
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
\( O(n^2) \) selection sort, bubble sort

\( O(n \log n) \)

\( O(n) \) linear search

\( O(\log n) \) binary search

\( O(1) \)
\( O(n^2) \) selection sort, bubble sort
\( O(n \log n) \) merge sort
\( O(n) \) linear search
\( O(\log n) \) binary search
\( O(1) \)
\(\Omega(n^2)\) selection sort

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

\(\Omega(n)\) bubble sort

\(\Omega(\log n)\)

\(\Omega(1)\) linear search, binary search
\( \Omega(n^2) \) selection sort

\( \Omega(n \log n) \) merge sort

\( \Omega(n) \) bubble sort

\( \Omega(\log n) \)

\( \Omega(1) \) linear search, binary search
\(\Theta(n^2)\)

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

\(\Theta(n)\)

\(\Theta(\log n)\)

\(\Theta(1)\)
\( \Theta(n^2) \) selection sort

\( \Theta(n \log n) \) merge sort

\( \Theta(n) \)

\( \Theta(\log n) \)

\( \Theta(1) \)
This is CS50