This is CS50
learn how to think algorithmically
The graph compares the time to solve different functions of the size of the problem: $n$, $n/2$, and $\log_2 n$. The time to solve increases significantly faster for $n$ and $n/2$ than for $\log_2 n$. This indicates that algorithms with $\log_2 n$ complexity are more efficient for large problem sizes.
1. Stand up and think of the number 1.
1. Stand up and think of the number 1.
2. Pair off with someone standing, add their number to yours, and remember the sum.
1. Stand up and think of the number 1.
2. Pair off with someone standing, add their number to yours, and remember the sum.
3. One of you should then sit down.
1. Stand up and think of the number 1.
2. Pair off with someone standing, add their number to yours, and remember the sum.
3. One of you should then sit down.
4. If still standing, go back to step 2.
Contacts

Search

B
Bowser
Bowser Jr.

D
Daisy
Diddy Kong
Donkey Kong

L
Luigi

M
Mario
John Harvard

- Mobile: +1 (949) 468-2750

Notes

Send Message
Share Contact
Add to Favorites
Add to Emergency Contacts
searching
input → 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 is given by:

- \(O(n)\) for the red line
- \(O(n/2)\) for the yellow line
- \(O(\log_2 n)\) for the green line
The time to solve a problem is given by different Big O notations:

- \( O(n) \) for linear time complexity.
- \( O(n/2) \) for time complexity that decreases linearly with the problem size.
- \( O(\log_2 n) \) for logarithmic time complexity.
The time to solve a problem is given by the following expressions:

- $O(n)$
- $O(n)$ for two scenarios
- $O(\log_2 n)$

These expressions indicate how the time to solve a problem grows with the size of the problem.
\begin{align*}
\text{time to solve} & : O(n) \quad O(n) \\
\text{size of problem} & : O(\log n)
\end{align*}
The diagram illustrates the relationship between the size of a problem and the time it takes to solve it. The graph shows two different functions: $O(n)$ and $O(\log n)$. The function $O(n)$ represents a linear relationship, where the time to solve increases proportionally with the size of the problem. On the other hand, $O(\log n)$ indicates a logarithmic relationship, where the time to solve increases much more slowly as the size of the problem grows.
$O(n^2)$

$O(n \log n)$

$O(n)$

$O(\log n)$

$O(1)$
\( O(n^2) \)
\( O(n \log n) \)
\( O(n) \)  
linear search
\( O(\log n) \)  
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
string.h
manual.cs50.io/#string.h
strcmp
data structures
person people[]
string name;
string number;
typedef struct
{
    string name;
    string number;
}
person;
typedef struct
{
    string name;
    string number;
} person;
sorting
input → output
unsorted $\rightarrow$ | $\rightarrow$ output
unsorted → sorted
7 2 5 4 1 6 0 3 → sorted
selection sort
For i from 0 to n-1
  Find smallest number between numbers[i] and numbers[n-1]
  Swap smallest number with numbers[i]
$(n - 1)$
(n - 1) + (n - 2)
\((n - 1) + (n - 2) + (n - 3)\)
\[(n - 1) + (n - 2) + (n - 3) + \ldots + 1\]
\[(n - 1) + (n - 2) + (n - 3) + \ldots + 1\]

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

\(n(n - 1)/2\)

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

\(n(n - 1)/2\)

\((n^2 - n)/2\)

\(n^2/2 - n/2\)

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

$\Omega(n \log n)$

$\Omega(n)$

$\Omega(\log n)$

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

$\Theta(n \log n)$

$\Theta(n)$

$\Theta(\log n)$

$\Theta(1)$
\( \Theta(n^2) \) 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
\((n - 1) \times (n - 1)\)
\[(n - 1) \times (n - 1)\]

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

$n^2 - 1n - 1n + 1$

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

$n^2 - 1n - 1n + 1$

$n^2 - 2n + 1$

$O(n^2)$
$O(n^2)$

$O(n \log n)$

$O(n)$

$O(\log n)$

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

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

\(\Omega(n)\)

\(\Omega(\log n)\)

\(\Omega(1)\)

bubble sort
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
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
5. Else if person is earlier in book
   6. Open to middle of left half of book
   7. Go back to line 3
6. Else if person is later in book
   7. Open to middle of right half of book
   8. Go back to line 3
7. Else
8. 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. Else if person is later in book
9.   Search right half of book
10. Else
11.   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
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
5270

41

36
\( O(n^2) \)
\( O(n \log n) \)
\( O(n) \)
\( O(\log n) \)
\( O(1) \)
$\log_2 n$
\[ \log_2 8 \]
\log_2 2^3
\( n \log_2 n \)
n \log n
$O(n^2)$

$O(n \log n)$  merge sort

$O(n)$

$O(\log n)$

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

$\Omega(n \log n)$

merge sort

$\Omega(n)$

$\Omega(\log n)$

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

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

\( \Theta(n) \)

\( \Theta(\log n) \)

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