Lecture 8 - Searching

Computer systems, data structures, and data management (4CM508)

Dr Sam O'Neill

Activity 1

In pairs (or threes) discuss the following.

- There are 7 cards face down.
- Turning over a card costs £1.

1. How much will it cost to **guarantee** whether the **Ace of Spades** is one of the 7 cards?
2. What about if we had 30 cards?

Linear Search

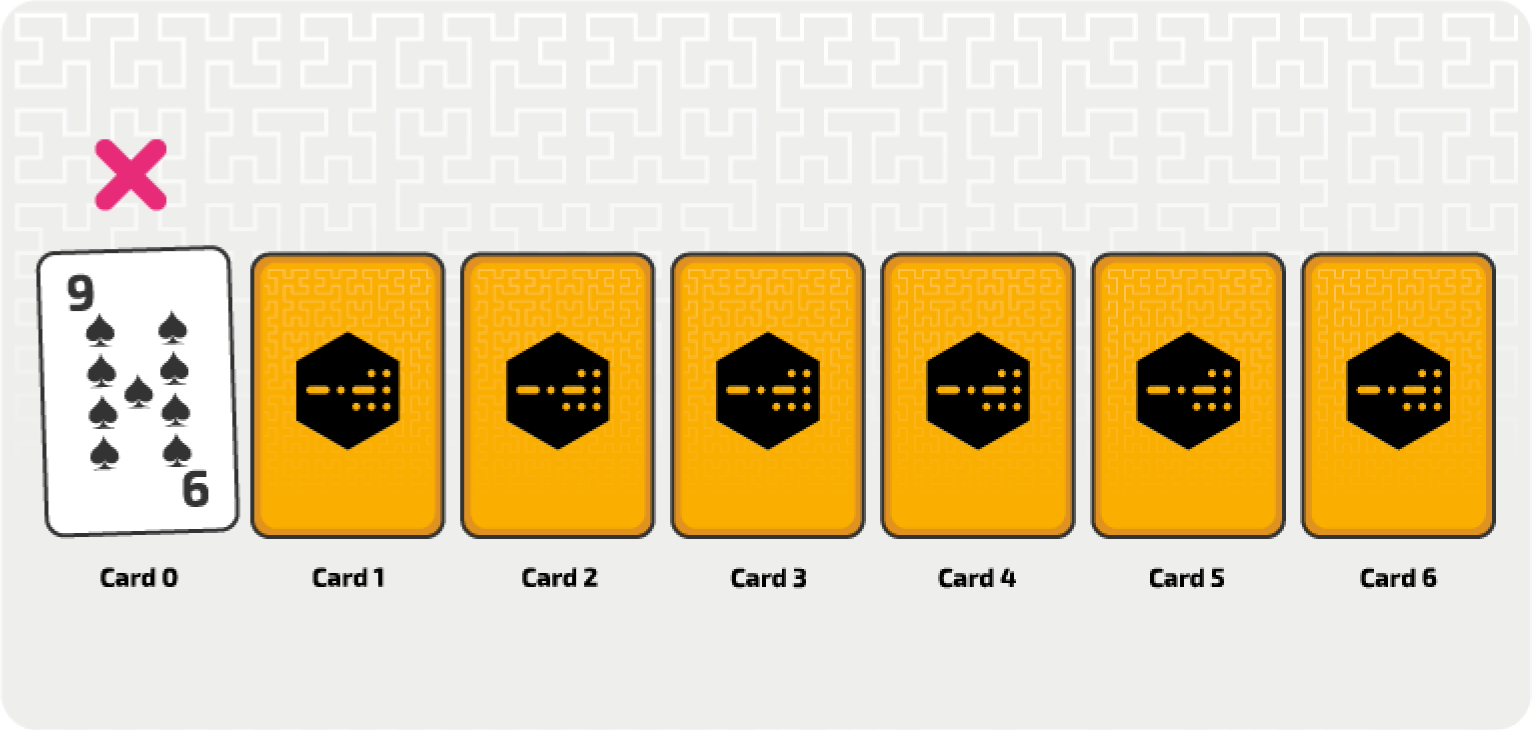
The method we just discussed is called Linear Search

The steps of linear search

The steps for performing a linear search can be described as follows:

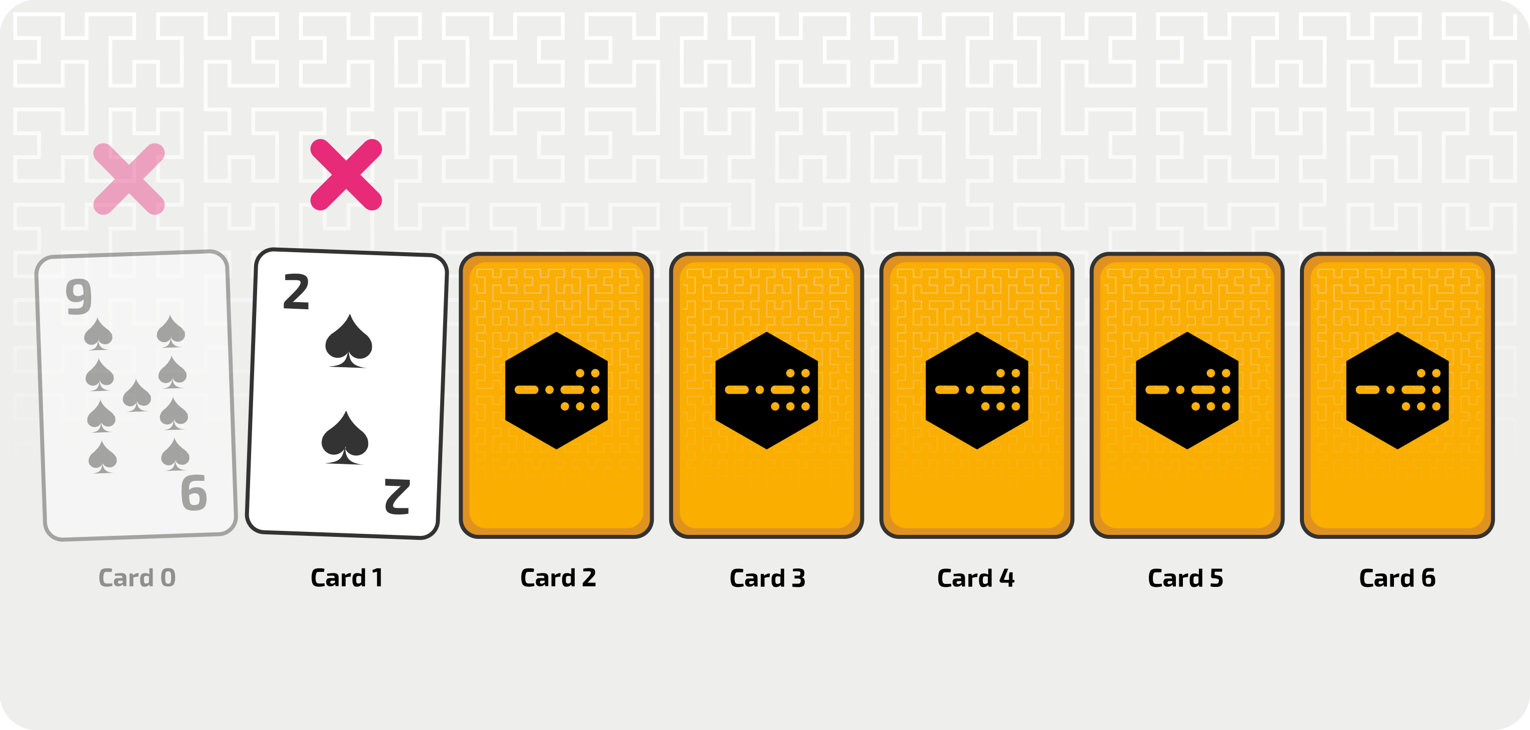
- **Step 1:** Take a list of data and an item that is being searched for (the search item)
- **Step 2:** Starting from the first position in the list, repeat steps 3–5 until you find the search item or until the end of the list is reached:
- **Step 3:** Compare the item at the current position in the list (index) to the search item
- **Step 4:** If the item at the current position (index) is equal to the search item, then stop searching
- **Step 5:** Otherwise, go to the next position in the list

Searching for the 4 Spades



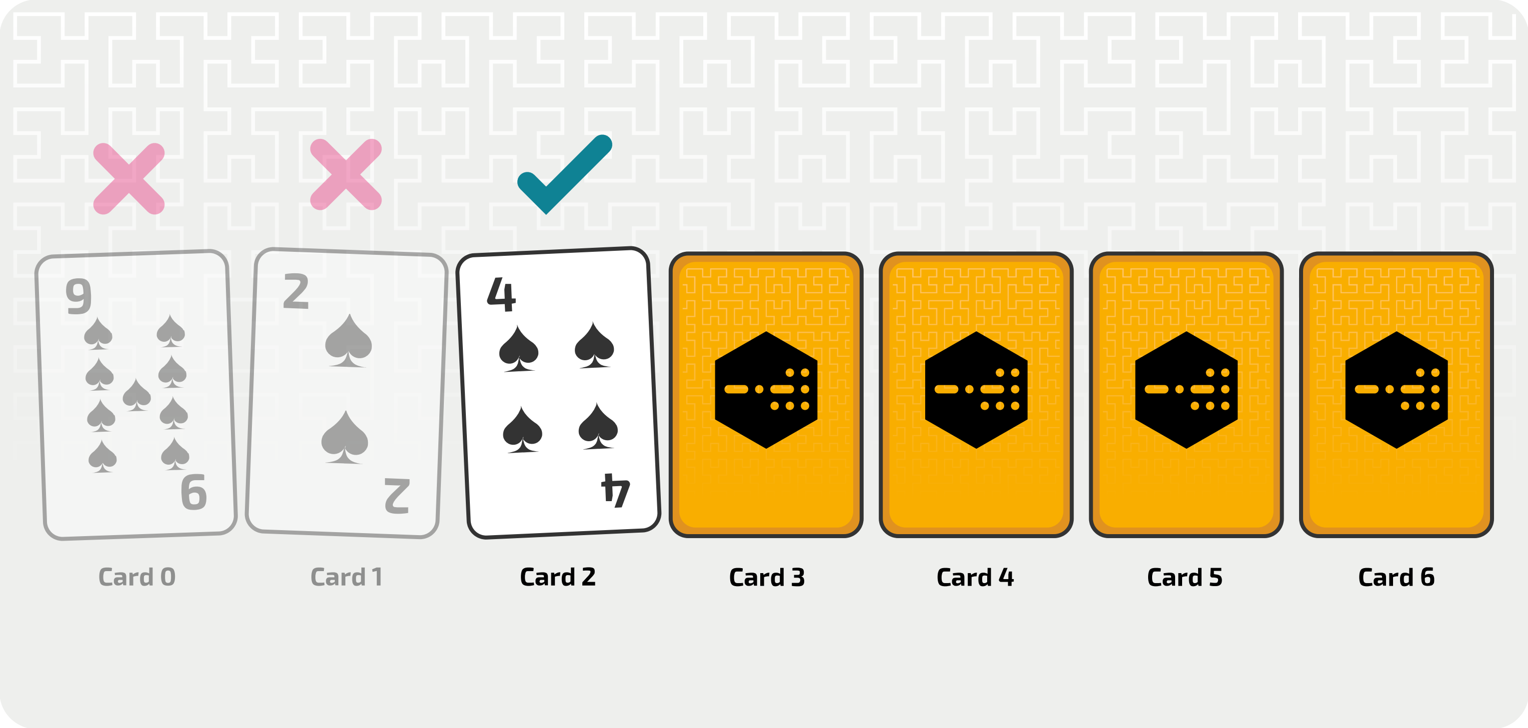
Check first card

Searching for the 4 Spades



Check second card

Searching for the 4 Spades



Check third card

Time Complexity

The worse-case is that the card isn't in the set of cards, thus we check every card.

For n cards this is $O(n)$.

In the best-case it is the first card, this is $O(1)$.

Space Complexity

Irrespective of the size (n) of list we are searching, linear search just requires a few variables that keep being overwritten.

The size taken is always the same, it doesn't depend on n.

Thus it is $O(1)$.

Python Implementation

Let's code this in Python.

You can find this with the reading.

Activity 2

In pairs (or threes) discuss the following.

- There are 7 cards face down
- Same suit (hearts)
- Cards in order - 1 (Ace), 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 (Jack), 12 (Queen), 13 (King)
- Turning over a card costs £1.

1. How much will it cost to **guarantee** whether the **7 of Hearts** is one of the 7 cards?
2. What if we had a list of 100 numbers and wanted to find 7?

Binary Search

The method we just discussed is called Binary Search

Binary Search Steps

- **Step 1:** Set the search range to be the entire list of ordered items.
- **Step 2:** Repeat steps 3–6 until you find the search item or there are no more items to check (the range is empty):
- **Step 3:** Find the item at the midpoint position (in the middle of the range).
- **Step 4:** Compare the item at the midpoint position to the search item.
- **Step 5:** If the item at the midpoint is equal to the search item, then stop searching.
- **Step 6:** Otherwise,
  - If the midpoint item $<$ search item, change the range to focus on the items after the midpoint. ***i.e. search item cannot be in items before midpoint***
  - if the midpoint item $>$ search item, change the range to focus on the items before the midpoint. ***i.e. search item cannot be in items after midpoint***

Binary Search Example

- Search for 25
- Set the whole list to the search range

items							
index	0	1	2	3	4	5	6
value	10	11	13	15	18	25	29
	-	-	-	-	-	-	-

Binary Search Example

- Search for 25
- Midpoint is 15. As 25 is greater than 15 we can discard the first 4 items.

items							
index	0	1	2	3	4	5	6
value	10	11	13	15	18	25	29
	-	-	-	midpoint	-	-	-

Binary Search Example

- Search for 25
- New search range

items							
index	0	1	2	3	4	5	6
value	10	11	13	15	18	25	29
					-	-	-

Binary Search Example

- Search for 25
- Midpoint is 25. We have found our item

items							
index	0	1	2	3	4	5	6
value	10	11	13	15	18	25	29
					-	midpoint	-

Binary Search Flowchart (iterative)

By iterative we mean using a loop

Let's draw this as a flowchart!

You can find this with the reading.

Binary Search Example (animation)

Let's see this as an animation!

You can find this with the reading.

YOU WILL FIND THIS VERY USEFUL!

Python Implementation (iterative)

Let's code this in Python.

You can find this with the reading.

Trace Table

Let's use Python Tutor to do a Trace Table.

We will fill out the following table. You can find an example in the reading.

Note: we may need more rows, this is just the layout.

`found_index`	`first`	`last`	`midpoint`	`items[midpoint]`	`found`	Return value

Space Complexity (iterative)

Irrespective of the size (n) of list we are searching, binary search just requires a few variables that keep being overwritten.

The size taken is always the same, it doesn't depend on n.

Thus it is $O(1)$.

Time Complexity (iterative)

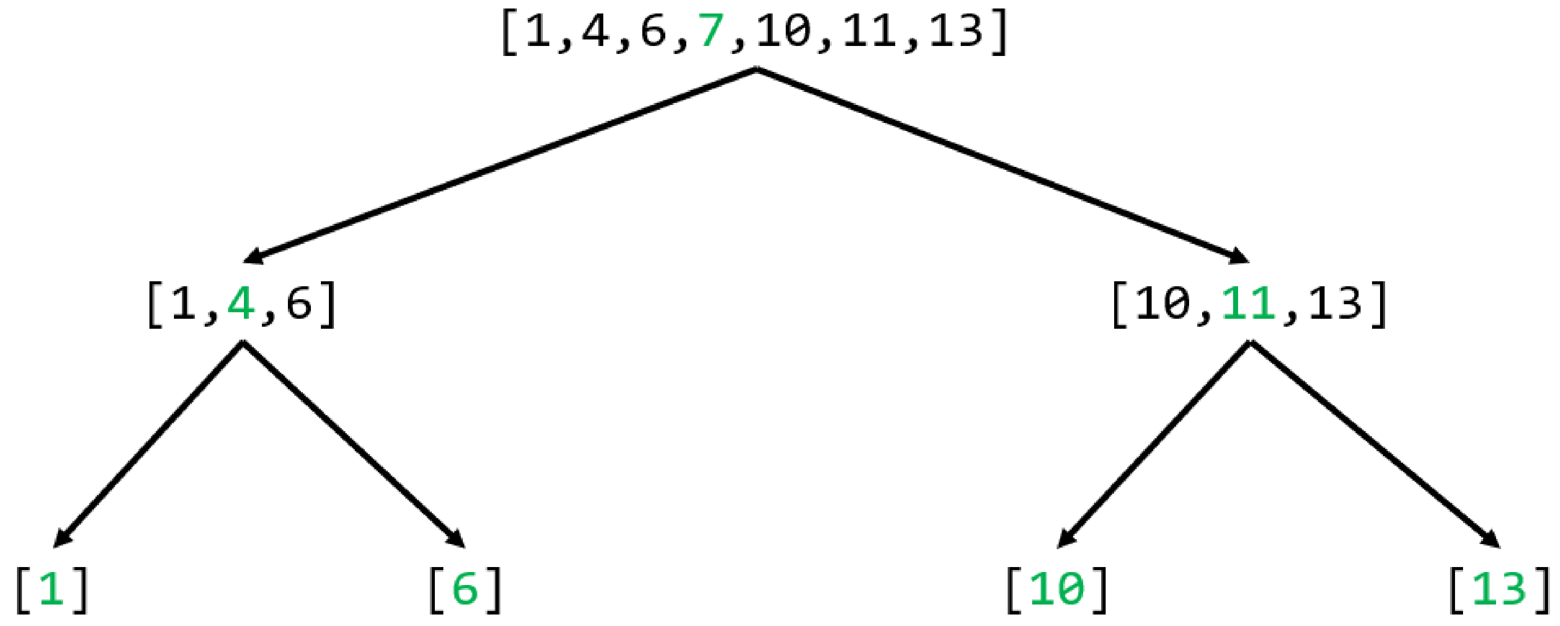
The best-case time complexity is just finding the search item on the first comparison - $O(1)$.

What is the worse-case time complexity?

In the worse-case we do not find the search item.

To work this out we need to examine some examples.

Example 1

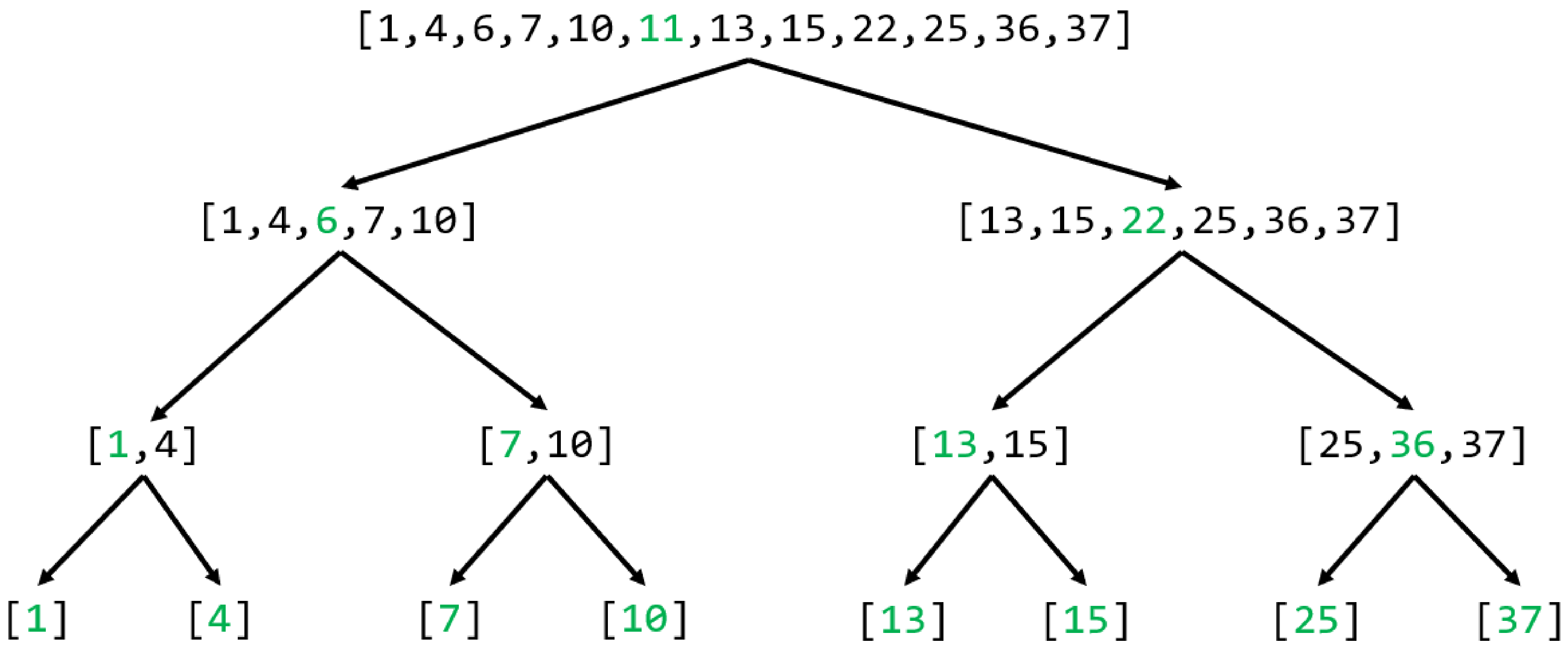


Tracing the number of midpoints checked.

Here the number of items in the list is $n = 7$

The max number of checks of the midpoint is $\lfloor \log(7) \rfloor + 1 = 3$

Example 2



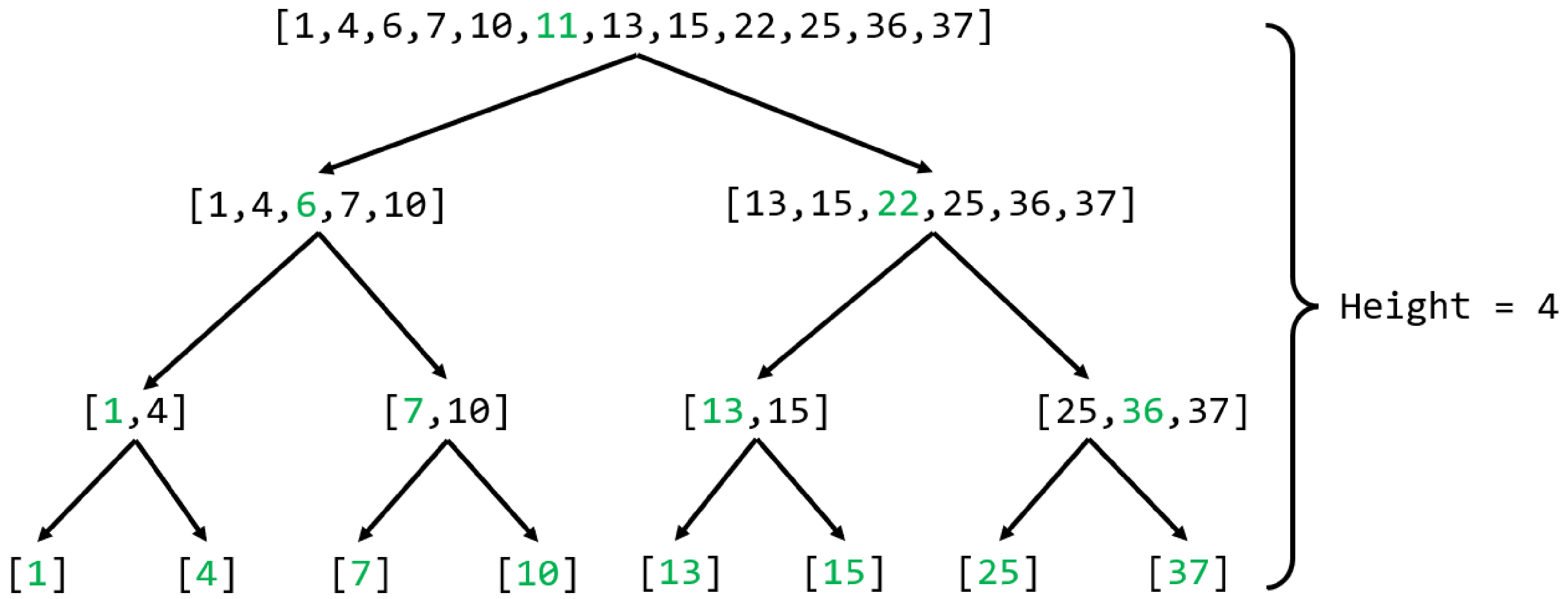
Tracing the number of midpoints checked.
Here the number of items in the list is $n = 12$

The max number of checks of the midpoint is $\lfloor \log(12) \rfloor + 1 = 4$

Comparison Table

Items	No. of items - n	Worse-case No. of iterations	$\log(n)$
`[1,4,6,7,10,11,13]`	`7`	`3`	`2.8074`
`[1,4,6,7,10,11,13,15,22,25,36,37]`	`12`	`4`	`3.585`
`[1,4,6,7,10,11,13,15,22,25,36,37, ...]`	`20`	`5`	`4.322`
`[1,4,6,7,10,11,13,15,22,25,36,37, ...]`	`32`	`6`	`5`
`[1,4,6,7,10,11,13,15,22,25,36,37, ...]`	`64`	`7`	`6`

Binary Tree Intuition



Number of items $n = 12$
Height = $\lfloor \log(12) \rfloor + 1 = 4$

A list with n items iterates $\lfloor log(n) \rfloor + 1$ times.

- $\lfloor \cdot \rfloor$ means the floor function. e.g. $\lfloor 5.3 \rfloor = 5$.
- $log(n)$ denotes base 2, i.e. $log_2(n)$

e.g. for 12 items.

$\lfloor log(12) \rfloor + 1 = \lfloor 3.585 \rfloor + 1 = 3 + 1 = 4$

We iterate 4 times.

We don't count the 1 for the worse-case complexity. Thus it is $O(log(n))$.

Better than linear search $O(\log(n)) < O(n)$.

Binary Search (recursive)

We can also do this using recursion. Please see the notes.

Note that this will have a space complexity of $O(\log(n))$.

Summary

- Linear search loops over all items.
  - Does not require **ordered** data
  - Worse-case time complexity - $O(n)$
  - Best-case time complexity - $O(1)$
  - Space complexity - $O(1)$
- Binary search (iterative) halves (approximately) the list each time.
  - Requires **ordered** data.
  - Worse-case time complexity - $O(log(n))$
  - Best-case time complexity - $O(1)$
  - Space complexity - $O(1)$