Binary Search

Introduction

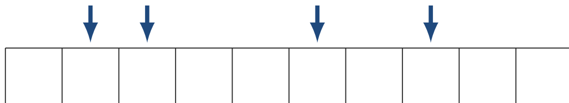
Problem

Problem

Given a dictionary and a word. Which page (if any) contains the given word?

Strategy 1: Random Search

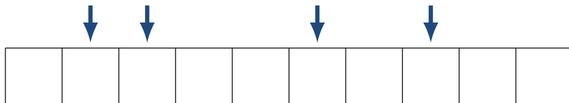
Randomly select a page until the page containing the word is found.



Strategy 1: Random Search

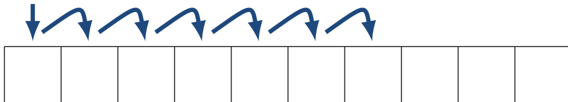
Randomly select a page until the page containing the word is found.

- Very inefficient, no guarantee that the correct page is chosen.
- Does not detect if word is not in the dictionary.



Strategy 2: Linear Search

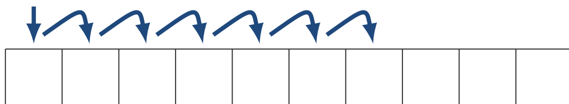
Check all pages sequentially starting at page 1.



Strategy 2: Linear Search

Check all pages sequentially starting at page 1.

- Acceptable efficiency, but ...
- strategy is not using all known properties.



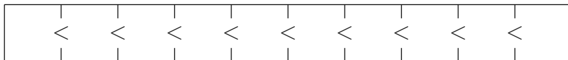
Strategy 3: Binary Search

Question: What is the main property of words in a dictionary?

Strategy 3: Binary Search

Question: What is the main property of words in a dictionary?

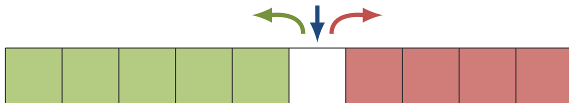
- Words are sorted.



Strategy 3: Binary Search

Question: What is the main property of words in a dictionary?

- Words are sorted.
- After looking on a page, we can say if word is on an earlier or later page.

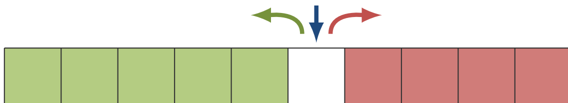


Strategy 3: Binary Search

Question: What is the main property of words in a dictionary?

- Words are sorted.
- After looking on a page, we can say if word is on an earlier or later page.

Select a page. If the word is smaller, look on an earlier page. If the word is larger, look on a later page.

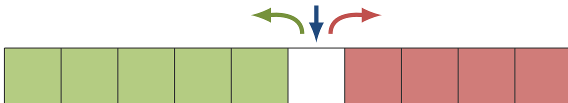


Strategy 3: Binary Search

Question: What is the main property of words in a dictionary?

- Words are sorted.
- After looking on a page, we can say if word is on an earlier or later page.

Select a page. If the word is smaller, look on an earlier page. If the word is larger, look on a later page.



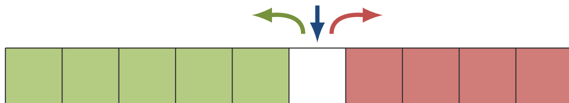
Question: Which page should be selected?

Strategy 3: Binary Search

Question: What is the main property of words in a dictionary?

- Words are sorted.
- After looking on a page, we can say if word is on an earlier or later page.

Select a page. If the word is smaller, look on an earlier page. If the word is larger, look on a later page.



Question: Which page should be selected?

- The middle!? Why?

Algorithm

Input:

- A sorted array A
- Some value k

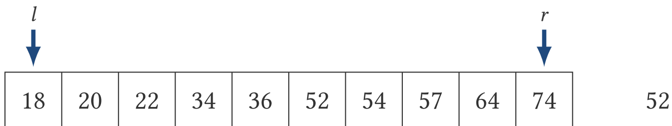
18	20	22	34	36	52	54	57	64	74

52

Algorithm

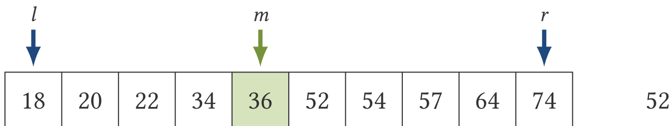
1. Set $l := 0$ and $r := A.\text{Length} - 1$.

2. **While** $l \leq r$



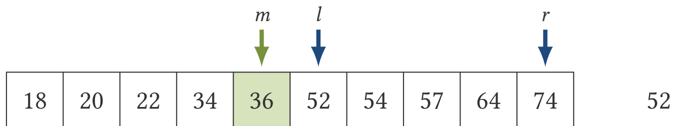
Algorithm

2.1. Set $m := \left\lfloor \frac{l+r}{2} \right\rfloor$.



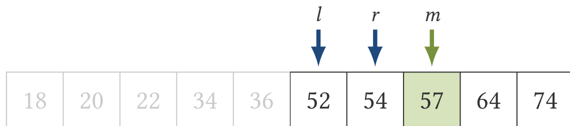
Algorithm

2.2. **If** $A[m] < k$ **Then** Set $l := m + 1$.

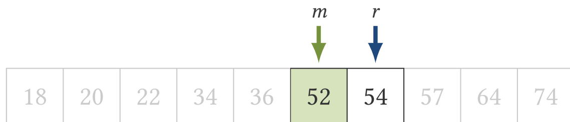


Algorithm

2.3. **If** $A[m] > k$ **Then** Set $r := m - 1$.



2.3. **IF** $A[m] = k$ **THEN RETURN** m.



Algorithm – BINARY SEARCH

Input:

- A sorted array A
- Some value k

1. Set $l := 0$ and $r := A.\text{Length} - 1$.
2. **WHILE** $l \leq r$
   2.1 Set $m := \left\lfloor \frac{l+r}{2} \right\rfloor$.
   2.2 **IF** $A[m] < k$ **THEN** Set $l := m + 1$.
   2.3 **IF** $A[m] > k$ **THEN** Set $r := m - 1$.
   2.4 **IF** $A[m] = k$ **THEN RETURN** m.
3. **RETURN** -1.

Correctness and Invariant

Correctness

What means BINARY SEARCH is correct?

Correctness

What means BINARY SEARCH is correct?

Theorem

The algorithm BINARY SEARCH returns the index of k in A if $k \in A$, otherwise it returns -1.

Correctness

What means BINARY SEARCH is correct?

Theorem

The algorithm BINARY SEARCH returns the index of k in A if $k \in A$, otherwise it returns -1.

Proof:
We have to show two cases

- $k \in A$
- $k \notin A$

We will show the second case first.

Proof of Correctness – $k \notin A$

Some notation:

- l_i and r_i denote the value of l and r at the beginning of the i-th loop iteration, i. e., $l_0 = 0$ and $r_0 = A$.Length $- 1$.
- m_i denotes the value of m assigned in the i-th loop iteration.

Proof of Correctness – $k \notin A$

Some notation:

- l_i and r_i denote the value of l and r at the beginning of the i-th loop iteration, i. e., $l_0 = 0$ and $r_0 = A.\text{Length} - 1$.
- m_i denotes the value of m assigned in the i-th loop iteration.

We have to show that we reach line 3.

- Because $k \notin A$, we never run line 2.4.
- Because $l_i \leq m_i \leq r_i$, $r_{i+1} - l_{i+1} < r_i - l_i$ (line 2.2 and line 2.3).

Thus, the algorithm reaches line 3 after a finite amount of iterations. $\square$

Proof of Correctness – $k \in A$

How do we show that we find the right index?

Proof of Correctness – $k \in A$

How do we show that we find the right index?

- We show an invariant.

Proof of Correctness – $k \in A$

How do we show that we find the right index?

- We show an invariant.

Invariant

The *invariant* of a loop is a property that is true every time before and after every iteration of the loop.

Proof of Correctness – $k \in A$

How do we show that we find the right index?

- We show an invariant.

Invariant

The *invariant* of a loop is a property that is true every time before and after every iteration of the loop.

Invariants often lead to the correctness of an algorithm.

The most common way to prove them is induction.

Proof of Correctness – $k \in A$

Invariant for Binary Search

Proof of Correctness – $k \in A$

Invariant for Binary Search

- If $k \in A$, then $l_i \leq \mathrm{Ind}(k) \leq r_i$. ($\mathrm{Ind}(k)$ is the index of k in A)
- $r_{i+1} - l_{i+1} < r_i - l_i$

Proof of Correctness – $k \in A$

Invariant for Binary Search

- If $k \in A$, then $l_i \leq \mathrm{Ind}(k) \leq r_i$. ($\mathrm{Ind}(k)$ is the index of k in A)
- $r_{i+1} - l_{i+1} < r_i - l_i$

Base Case

- Is true, because $l_0 = 0$ and $r_0 = A$.Length $- 1$.

Proof of Correctness – $k \in A$

Invariant for Binary Search

- If $k \in A$, then $l_i \leq \text{Ind}(k) \leq r_i$. ($\text{Ind}(k)$ is the index of k in A)
- $r_{i+1} - l_{i+1} < r_i - l_i$

Base Case

- Is true, because $l_0 = 0$ and $r_0 = A.\text{Length} - 1$.

Inductive Step

- By induction, $l_i \leq \text{Ind}(k) \leq r_i$.
- If $k = A[m_i]$, the algorithm returns m_i. Done.
- If $k < A[m_i]$, then $\text{Ind}(k) < m_i$.
  Thus, $l_{i+1} = l_i \leq \text{Ind}(k) \leq r_{i+1} = m_i - 1 < m_i \leq r_i$.
- If $k > A[m_i]$, then $\text{Ind}(k) > m_i$.
  Thus, $l_i \leq m_i < m_i + 1 = l_{i+1} \leq \text{Ind}(k) \leq r_{i+1} = r_i$.

Proof of Correctness – $k \in A$

Invariant for Binary Search

- If $k \in A$, then $l_i \leq \text{Ind}(k) \leq r_i$. ($\text{Ind}(k)$ is the index of k in A)
- $r_{i+1} - l_{i+1} < r_i - l_i$

Base Case

- Is true, because $l_0 = 0$ and $r_0 = A.\text{Length} - 1$.

Inductive Step

- By induction, $l_i \leq \text{Ind}(k) \leq r_i$.
- If $k = A[m_i]$, the algorithm returns m_i. Done.
- If $k < A[m_i]$, then $\text{Ind}(k) < m_i$.
  Thus, $l_{i+1} = l_i \leq \text{Ind}(k) \leq r_{i+1} = m_i - 1 < m_i \leq r_i$.
- If $k > A[m_i]$, then $\text{Ind}(k) > m_i$.
  Thus, $l_i \leq m_i < m_i + 1 = l_{i+1} \leq \text{Ind}(k) \leq r_{i+1} = r_i$.

Because $r_{i+1} - l_{i+1} < r_i - l_i$, there is an iteration j with $l_j = \text{Ind}(k) = r_j$. $\square$

Runtime

Runtime

How fast is Binary Search?

Runtime

How fast is BINARY SEARCH?

- We count number of iterations.
- $r_0 - l_0 = n - 1$
- $r_{i+1} - l_{i+1} \leq (r_i - l_i)/2$

Runtime

How fast is BINARY SEARCH?

- We count number of iterations.
- $r_0 - l_0 = n - 1$
- $r_{i+1} - l_{i+1} \leq (r_i - l_i)/2$

For which j is $r_j - l_j \leq 1$?

Runtime

How fast is BINARY SEARCH?

- We count number of iterations.
- $r_0 - l_0 = n - 1$
- $r_{i+1} - l_{i+1} \leq (r_i - l_i)/2$

For which j is $r_j - l_j \leq 1$?

$$
\begin{aligned}
r_j - l_j &\leq \frac{1}{2}(r_{j-1} - l_{j-1}) \\
&\leq \frac{1}{4}(r_{j-2} - l_{j-2}) \\
&\vdots \\
&\leq 2^{-j}(r_0 - l_0) = 2^{-j}(n-1) \leq 1
\end{aligned}
$$

$$\log_2 n \leq j$$

Runtime

Theorem

The algorithm BINARY SEARCH requires at most $\lceil \log_2(n+1) \rceil$ iterations, where $n = |A|$.

Logarithmic Runtime

If $|A| = 2^n$ we need approx. n iterations.

Also, $10^3 \approx 2^{10}$

Input size – n	**Runtime – $\log_2 n$**
1,000	10
1,000,000	20
1,000,000,000	30
atoms in the universe	266

Lower Time Bound

Is there a faster way than BINARY SEARCH?

- We allow a *free* preprocessing of A.
- We want to know if k is in A and where.

Lower Time Bound

Is there a faster way than BINARY SEARCH?

- We allow a *free* preprocessing of A.
- We want to know if k is in A and where.

Theorem

In general, finding an element in an array cannot be done in less than logarithmic time.

Decision Tree

Model of computation: comparison model

- only operations allowed are comparisons ($<, \leq, >, \geq, =, \neq$)
- time cost is number of comparisons

Decision Tree

Model of computation: comparison model

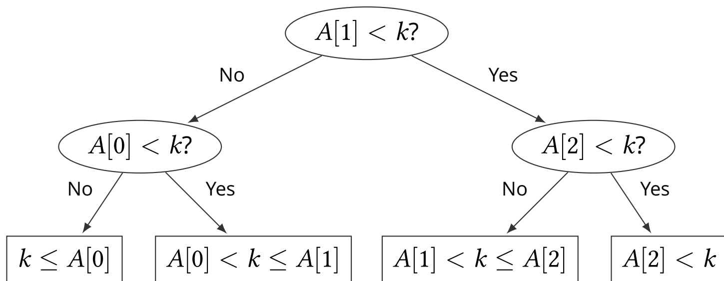
- only operations allowed are comparisons ($<, \leq, >, \geq, =, \neq$)
- time cost is number of comparisons

Decision Tree

- algorithm is tree of comparisons
- internal nodes are comparisons
- leafs are outcomes of the algorithm

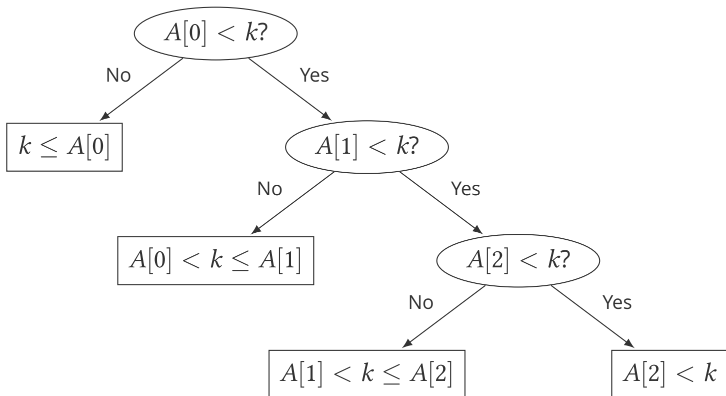
Decision Tree Example

BINARY SEARCH for $|A| = 3$
(For simplicity we ignore the case $A[i] = k$.)



Decision Tree Example

LINEAR SEARCH for $|A| = 3$
(For simplicity we ignore the case $A[i] = k$.)



Decision Tree vs. Algorithm + Proof

Decision Tree	Algorithm
internal node	binary decision
leaf	found answer
root-to-leaf path	algorithm execution
path length	running time
height of tree	worst case running time

Decision Tree vs. Algorithm + Proof

Decision Tree	Algorithm
internal node	binary decision
leaf	found answer
root-to-leaf path	algorithm execution
path length	running time
height of tree	worst case running time

Proof of theorem:

- Decision tree is binary
- Tree contains each possible answer as leaf, i. e., $n + 1$ leafs
- Height $\geq \log_2(\text{number of leafs})$.

Exercises

Customer Database

A company database consists of 10,000 sorted names, 40 % of whom are known as good customers and who together account for 60 % of the accesses to the database. There are two data structure options to consider for representing the database:

(1) Put all the names in a single array and use binary search.

(2) Put the good customers in one array and the rest of them in a second array.

Only if we do not find the query name on a binary search of the first array, we do a binary search of the second array.

Demonstrate which option gives better expected performance. Does this change if linear search on an unsorted array is used instead of binary search for both options?

Customer Database — Using Binary Search

Option 1. A single array.

- Runtime (iterations): $\log n$

Option 2. Two arrays.

- Runtime:

$$\begin{aligned} &\log(0.4n) + 0.4\log(0.6n) \\ &= \log 0.4 + \log n + 0.4\log 0.6 + 0.4\log n \\ &= 1.4\log n - 1.6 \end{aligned}$$

Conclusion. It is better to keep all customers in a single array.

Customer Database — Using Linear Search

Option 1. A single array.

- Runtime (iterations): n

Option 2. Two arrays.

- Runtime:

$$
\begin{aligned}
&0.4n + 0.4 \cdot 0.6n \\
&= 0.64n
\end{aligned}
$$

Conclusion. It is better to split customers into two arrays.

Finding 01 in a Bit Array

You are given a bit array A where $A[0] = 0$ and $A[n-1] = 1$. Find an i such that $A[i] = 0$ and $A[i+1] = 1$.

0	$\cdots$	0	1	$\cdots$	1

Finding 01 in a Bit Array

You are given a bit array A where $A[0] = 0$ and $A[n-1] = 1$. Find an i such that $A[i] = 0$ and $A[i+1] = 1$.

Using binary search:

- If $A[m] = 1$, search left. If $A[m] = 0$, search right.
- Invariant: $A[l] = 0$ and $A[r] = 1$.

0	$\cdots$	0	1	$\cdots$	1

Nodes in a Complete Binary Tree

Complete binary tree

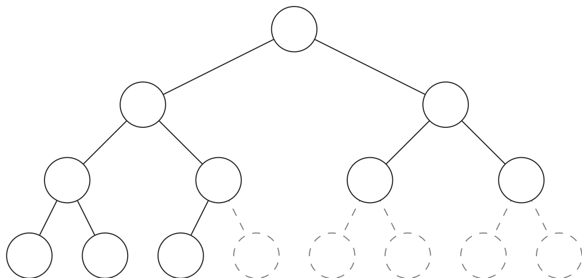
- All layers (except lowest) are full.
- Lowest layer is filled from left to right.

Question: How many nodes are in the tree?

Nodes in a Complete Binary Tree

Naive solution

- Count all nodes.
- Runtime: n



Nodes in a Complete Binary Tree

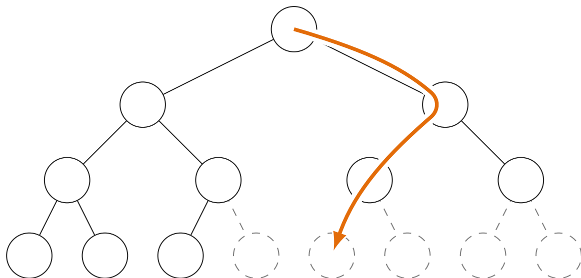
Using binary search

- Determine height h_l of left subtree (left side).

Nodes in a Complete Binary Tree

Using binary search

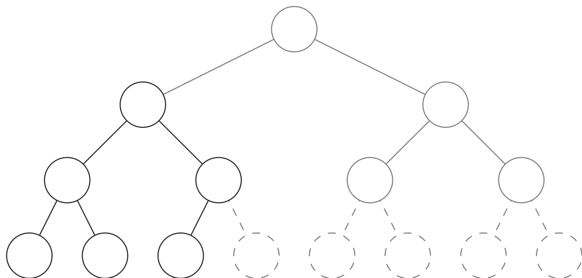
- Determine height h_l of left subtree (left side).
- Determine height h_r of right subtree (left side).



Nodes in a Complete Binary Tree

Using binary search

- Determine height h_l of left subtree (left side).
- Determine height h_r of right subtree (left side).
- If $h_l = h_r$, lowest layer of left subtree is full (continue with right subtree). Otherwise, lowest layer of right subtree is empty (continue with left subtree).
- Runtime: $\log^2 n$



Other

You are given a sorted array A of distinct integers. Determine whether there exists an index i such that $A[i] = i$.

You are given two sorted integer arrays A and B such that no integer is contained twice in the same array. A and B are nearly identical. However, B is missing one number. Determine the number missing in B.