

Linear and Binary Search

COMS10018 - Algorithms

Dr Christian Konrad

Runtime of Algorithms

Runtime of Algorithms

Consider an algorithm \mathcal{A} for a specific problem \mathcal{P}

Runtime of Algorithms

Consider an algorithm \mathcal{A} for a specific problem \mathcal{P}

Set of Potential Inputs

Consider an algorithm \mathcal{A} for a specific problem \mathcal{P}

Set of Potential Inputs

- Let $S(n)$ be the set of all potential inputs of length n for \mathcal{P}

Consider an algorithm \mathcal{A} for a specific problem \mathcal{P}

Set of Potential Inputs

- Let $S(n)$ be the set of all potential inputs of length n for \mathcal{P}
- For $X \in S(n)$, let $T(X)$ be the runtime of \mathcal{A} on input X

Runtime of Algorithms

Consider an algorithm \mathcal{A} for a specific problem \mathcal{P}

Set of Potential Inputs

- Let $S(n)$ be the set of all potential inputs of length n for \mathcal{P}
- For $X \in S(n)$, let $T(X)$ be the runtime of \mathcal{A} on input X

Worst-case Runtime: $\max_{X \in S(n)} T(X)$

Best-case Runtime: $\min_{X \in S(n)} T(X)$

Average-case Runtime: $\frac{1}{|S(n)|} \sum_{X \in S(n)} T(X)$

Linear Search:

- **Input:** Array A of n integers from range $\{0, 1, 2, \dots, k - 1\}$, for some integer k , integer $t \in \{0, 1, 2, \dots, k - 1\}$
- **Output:** 1, if A contains t , 0 otherwise

```
Require: Array  $A$ , integer  $t$   
  for  $i = 0, \dots, n - 1$  do  
    if  $A[i] = t$  then  
      return 1  
  return 0
```


Linear Search:

- **Input:** Array A of n integers from range $\{0, 1, 2, \dots, k - 1\}$, for some integer k , integer $t \in \{0, 1, 2, \dots, k - 1\}$
- **Output:** 1, if A contains t , 0 otherwise

Worst-case Runtime:

```
Require: Array  $A$ , integer  $t$   
for  $i = 0, \dots, n - 1$  do  
    if  $A[i] = t$  then  
        return 1  
return 0
```

Linear Search:

- **Input:** Array A of n integers from range $\{0, 1, 2, \dots, k - 1\}$, for some integer k , integer $t \in \{0, 1, 2, \dots, k - 1\}$
- **Output:** 1, if A contains t , 0 otherwise

Worst-case Runtime: $\Theta(n)$

```
Require: Array  $A$ , integer  $t$   
for  $i = 0, \dots, n - 1$  do  
    if  $A[i] = t$  then  
        return 1  
return 0
```

Linear Search:

- **Input:** Array A of n integers from range $\{0, 1, 2, \dots, k - 1\}$, for some integer k , integer $t \in \{0, 1, 2, \dots, k - 1\}$
- **Output:** 1, if A contains t , 0 otherwise

Worst-case Runtime: $\Theta(n)$

E.g. on any input with
 $A[i] \neq t$ for every i

```
Require: Array  $A$ , integer  $t$   
for  $i = 0, \dots, n - 1$  do  
    if  $A[i] = t$  then  
        return 1  
return 0
```

Linear Search

Linear Search:

- **Input:** Array A of n integers from range $\{0, 1, 2, \dots, k - 1\}$, for some integer k , integer $t \in \{0, 1, 2, \dots, k - 1\}$
- **Output:** 1, if A contains t , 0 otherwise

Worst-case Runtime: $\Theta(n)$

E.g. on any input with
 $A[i] \neq t$ for every i

Best-case Runtime:

```
Require: Array  $A$ , integer  $t$   
for  $i = 0, \dots, n - 1$  do  
    if  $A[i] = t$  then  
        return 1  
return 0
```

Linear Search

Linear Search:

- **Input:** Array A of n integers from range $\{0, 1, 2, \dots, k - 1\}$, for some integer k , integer $t \in \{0, 1, 2, \dots, k - 1\}$
- **Output:** 1, if A contains t , 0 otherwise

Worst-case Runtime: $\Theta(n)$

E.g. on any input with
 $A[i] \neq t$ for every i

Best-case Runtime: $O(1)$

```
Require: Array  $A$ , integer  $t$   
for  $i = 0, \dots, n - 1$  do  
    if  $A[i] = t$  then  
        return 1  
return 0
```

Linear Search

Linear Search:

- **Input:** Array A of n integers from range $\{0, 1, 2, \dots, k - 1\}$, for some integer k , integer $t \in \{0, 1, 2, \dots, k - 1\}$
- **Output:** 1, if A contains t , 0 otherwise

Worst-case Runtime: $\Theta(n)$

E.g. on any input with
 $A[i] \neq t$ for every i

Best-case Runtime: $O(1)$

On any input with $A[0] = t$

```
Require: Array  $A$ , integer  $t$   
for  $i = 0, \dots, n - 1$  do  
    if  $A[i] = t$  then  
        return 1  
return 0
```

Average-case Runtime: (over all possible inputs of length n)

Average-case Analysis of Linear Search

Possible Inputs of Length n

$$S(n) := \{\text{arrays } A \text{ of length } n \text{ with } A[i] \in \{0, 1, 2, \dots, k-1\}, \\ \text{for every } 0 \leq i \leq n-1\}$$

Average-case Analysis of Linear Search

Possible Inputs of Length n

$S(n) := \{\text{arrays } A \text{ of length } n \text{ with } A[i] \in \{0, 1, 2, \dots, k-1\},$
for every $0 \leq i \leq n-1\}$

$$|S(n)| = k^n.$$

Average-case Analysis of Linear Search

Possible Inputs of Length n

$S(n) := \{\text{arrays } A \text{ of length } n \text{ with } A[i] \in \{0, 1, 2, \dots, k-1\},$
for every $0 \leq i \leq n-1\}$

$$|S(n)| = k^n.$$

Auxiliary Function: For $A \in S(n)$, $t \in \{0, 1, \dots, k-1\}$:

$$\text{LEFT}(A, t) = \min\{i : A[i] = t\}.$$

If no such position exists then $\text{LEFT}(A, t) = n$.

Average-case Analysis of Linear Search

Possible Inputs of Length n

$S(n) := \{ \text{arrays } A \text{ of length } n \text{ with } A[i] \in \{0, 1, 2, \dots, k-1\},$
for every $0 \leq i \leq n-1 \}$

$$|S(n)| = k^n.$$

Auxiliary Function: For $A \in S(n)$, $t \in \{0, 1, \dots, k-1\}$:

$$\text{LEFT}(A, t) = \min\{i : A[i] = t\}.$$

If no such position exists then $\text{LEFT}(A, t) = n$.

Examples:

- $\text{LEFT}(23192, 9) = 3$
- $\text{LEFT}(0000, 1) = 4$

Average-case Analysis of Linear Search

Possible Inputs of Length n

$S(n) := \{\text{arrays } A \text{ of length } n \text{ with } A[i] \in \{0, 1, 2, \dots, k-1\},$
for every $0 \leq i \leq n-1\}$

$$|S(n)| = k^n.$$

Auxiliary Function: For $A \in S(n)$, $t \in \{0, 1, \dots, k-1\}$:

$$\text{LEFT}(A, t) = \min\{i : A[i] = t\}.$$

If no such position exists then $\text{LEFT}(A, t) = n$.

Examples:

- $\text{LEFT}(23192, 9) = 3$
- $\text{LEFT}(0000, 1) = 4$

→ Linear search loop executed $\text{LEFT}(X, t) + 1$ times

Average-case Analysis of Linear Search (continued)

Average-case Runtime for $k = 2$: (binary strings)

We compute average number of steps the loop is executed ($t = 1$)

$$\text{AVG} = \frac{1}{|S(n)|} \sum_{A \in S(n)} \text{LEFT}(A, 1) + 1$$

Average-case Analysis of Linear Search (continued)

Average-case Runtime for $k = 2$: (binary strings)

We compute average number of steps the loop is executed ($t = 1$)

$$\begin{aligned} \text{AVG} &= \frac{1}{|S(n)|} \sum_{A \in S(n)} \text{LEFT}(A, 1) + 1 \\ &= 2^{-n} \left(\left(\sum_{i=0}^{n-1} |\{A : \text{LEFT}(A, 1) = i\}| \cdot (i + 1) \right) + (n + 1) \right) . \end{aligned}$$

Average-case Analysis of Linear Search (continued)

Average-case Runtime for $k = 2$: (binary strings)

We compute average number of steps the loop is executed ($t = 1$)

$$\begin{aligned} \text{AVG} &= \frac{1}{|S(n)|} \sum_{A \in S(n)} \text{LEFT}(A, 1) + 1 \\ &= 2^{-n} \left(\left(\sum_{i=0}^{n-1} |\{A : \text{LEFT}(A, 1) = i\}| \cdot (i+1) \right) + (n+1) \right) . \end{aligned}$$

$\underbrace{0\ 0\ 0\ 0\ \dots\ 0}_{i \text{ times}}\ 1\ \underbrace{X\ X\ X\ \dots\ X}_{n-i-1 \text{ times}}$

Average-case Analysis of Linear Search (continued)

Average-case Runtime for $k = 2$: (binary strings)

We compute average number of steps the loop is executed ($t = 1$)

$$\begin{aligned}\text{AVG} &= \frac{1}{|S(n)|} \sum_{A \in S(n)} \text{LEFT}(A, 1) + 1 \\ &= 2^{-n} \left(\left(\sum_{i=0}^{n-1} |\{A : \text{LEFT}(A, 1) = i\}| \cdot (i+1) \right) + (n+1) \right) .\end{aligned}$$

$\underbrace{0\ 0\ 0\ 0\ \dots\ 0}_i\ 1\ \underbrace{X\ X\ X\ \dots\ X}_{n-i-1}$
 i times $n-i-1$ times

$$= 2^{-n} \left(\left(\sum_{i=0}^{n-1} 2^{n-1-i} \cdot (i+1) \right) + (n+1) \right)$$

$$= \left(\sum_{i=0}^{n-1} \frac{i+1}{2^{i+1}} \right) + (n+1)2^{-n}$$

Average-case Analysis of Linear Search (continued)

Average-case Runtime for $k = 2$: (binary strings)

We compute average number of steps the loop is executed ($t = 1$)

$$\begin{aligned} \text{AVG} &= \frac{1}{|S(n)|} \sum_{A \in S(n)} \text{LEFT}(A, 1) + 1 \\ &= 2^{-n} \left(\left(\sum_{i=0}^{n-1} |\{A : \text{LEFT}(A, 1) = i\}| \cdot (i+1) \right) + (n+1) \right) . \end{aligned}$$

$\underbrace{0\ 0\ 0\ 0\ \dots\ 0}_i\ 1\ \underbrace{X\ X\ X\ \dots\ X}_{n-i-1}$
 i times $n-i-1$ times

$$\begin{aligned} &= 2^{-n} \left(\left(\sum_{i=0}^{n-1} 2^{n-1-i} \cdot (i+1) \right) + (n+1) \right) \\ &= \left(\sum_{i=0}^{n-1} \frac{i+1}{2^{i+1}} \right) + (n+1)2^{-n} \leq 2 + 1 = 3 = O(1) . \end{aligned}$$

Average-case Analysis of Linear Search (continued)

Average-case Runtime for $k = 2$: (binary strings)

We compute average number of steps the loop is executed ($t = 1$)

$$\begin{aligned}\text{AVG} &= \frac{1}{|S(n)|} \sum_{A \in S(n)} \text{LEFT}(A, 1) + 1 \\ &= 2^{-n} \left(\left(\sum_{i=0}^{n-1} |\{A : \text{LEFT}(A, 1) = i\}| \cdot (i+1) \right) + (n+1) \right) .\end{aligned}$$

$\underbrace{0\ 0\ 0\ 0\ \dots\ 0}_i\ 1\ \underbrace{X\ X\ X\ \dots\ X}_{n-i-1}$
 i times $n-i-1$ times

$$= 2^{-n} \left(\left(\sum_{i=0}^{n-1} 2^{n-1-i} \cdot (i+1) \right) + (n+1) \right) \rightarrow \text{AVG-case runtime is } O(1)$$

$$= \left(\sum_{i=0}^{n-1} \frac{i+1}{2^{i+1}} \right) + (n+1)2^{-n} \leq 2 + 1 = 3 = O(1) .$$

(Trick for Bounding Sums)

How to bound $\sum_{i=0}^n \frac{i}{2^i}$:

$$S_n := \sum_{i=0}^n \frac{i}{2^i} .$$

(Trick for Bounding Sums)

How to bound $\sum_{i=0}^n \frac{i}{2^i}$:

$$S_n := \sum_{i=0}^n \frac{i}{2^i} .$$

Trick: Consider $\frac{1}{2}S_n$

(Trick for Bounding Sums)

How to bound $\sum_{i=0}^n \frac{i}{2^i}$:

$$S_n := \sum_{i=0}^n \frac{i}{2^i} .$$

Trick: Consider $\frac{1}{2}S_n$

$$S_n = \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + \cdots + \frac{n}{2^n}$$

(Trick for Bounding Sums)

How to bound $\sum_{i=0}^n \frac{i}{2^i}$:

$$S_n := \sum_{i=0}^n \frac{i}{2^i} .$$

Trick: Consider $\frac{1}{2}S_n$

$$\begin{aligned} S_n &= \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + \cdots + \frac{n}{2^n} \\ \frac{1}{2}S_n &= \frac{1}{4} + \frac{2}{8} + \frac{3}{16} + \cdots + \frac{n}{2^{n+1}} \end{aligned}$$

(Trick for Bounding Sums)

How to bound $\sum_{i=0}^n \frac{i}{2^i}$:

$$S_n := \sum_{i=0}^n \frac{i}{2^i} .$$

Trick: Consider $\frac{1}{2}S_n$

$$\begin{aligned} S_n &= \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + \cdots + \frac{n}{2^n} \\ \frac{1}{2}S_n &= \frac{1}{4} + \frac{2}{8} + \frac{3}{16} + \cdots + \frac{n}{2^{n+1}} \\ S_n - \frac{1}{2}S_n &= \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots + \frac{1}{2^n} - \frac{n}{2^{n+1}} \end{aligned}$$

(Trick for Bounding Sums)

How to bound $\sum_{i=0}^n \frac{i}{2^i}$:

$$S_n := \sum_{i=0}^n \frac{i}{2^i} .$$

Trick: Consider $\frac{1}{2}S_n$

$$\begin{aligned} S_n &= \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + \cdots + \frac{n}{2^n} \\ \frac{1}{2}S_n &= \frac{1}{4} + \frac{2}{8} + \frac{3}{16} + \cdots + \frac{n}{2^{n+1}} \\ S_n - \frac{1}{2}S_n &= \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots + \frac{1}{2^n} - \frac{n}{2^{n+1}} \\ &= \left(\sum_{i=1}^n \frac{1}{2^i} \right) - \frac{n}{2^{n+1}} \leq \frac{\frac{1}{2} - \frac{1}{2^{n+1}}}{1 - \frac{1}{2}} \leq 1 . \end{aligned}$$

$$\rightarrow S_n \leq 2$$

Binary Search:

- **Input:** A sorted array A of integers, an integer t
- **Output:** -1 if A does not contain t , otherwise a position i such that $A[i] = t$

Binary Search

Binary Search:

- **Input:** A sorted array A of integers, an integer t
- **Output:** -1 if A does not contain t , otherwise a position i such that $A[i] = t$

Require: Sorted array A of length n , integer t

if $|A| \leq 2$ **then**

 Check $A[0]$ and $A[1]$ and **return** answer

if $A[\lfloor n/2 \rfloor] = t$ **then**

return $\lfloor n/2 \rfloor$

else if $A[\lfloor n/2 \rfloor] > t$ **then**

return $\text{BINARY-SEARCH}(A[0, \dots, \lfloor n/2 \rfloor - 1])$

else

return $\lfloor n/2 \rfloor + 1 + \text{BINARY-SEARCH}(A[\lfloor n/2 \rfloor + 1, n - 1])$

Algorithm BINARY-SEARCH

Worst-case Analysis of Binary Search

Worst-case Analysis

Worst-case Analysis of Binary Search

Worst-case Analysis

- Without recursive calls, we spend $O(1)$ time in the function

Worst-case Analysis of Binary Search

Worst-case Analysis

- Without recursive calls, we spend $O(1)$ time in the function
- Worst-case runtime = $\underbrace{\text{"maximum \# of recursive calls"}}_r \cdot O(1)$

Worst-case Analysis of Binary Search

Worst-case Analysis

- Without recursive calls, we spend $O(1)$ time in the function
- Worst-case runtime = $\underbrace{\text{"maximum \# of recursive calls"}}_r \cdot O(1)$
- Observe that in iteration i the size of the array is at most half the size than in iteration $i - 1$

Worst-case Analysis of Binary Search

Worst-case Analysis

- Without recursive calls, we spend $O(1)$ time in the function
- Worst-case runtime = $\underbrace{\text{"maximum \# of recursive calls"}}_r \cdot O(1)$
- Observe that in iteration i the size of the array is at most half the size than in iteration $i - 1$
- We stop as soon as the size of the array is at most two

Worst-case Analysis of Binary Search

Worst-case Analysis

- Without recursive calls, we spend $O(1)$ time in the function
- Worst-case runtime = $\underbrace{\text{"maximum \# of recursive calls"}}_r \cdot O(1)$
- Observe that in iteration i the size of the array is at most half the size than in iteration $i - 1$
- We stop as soon as the size of the array is at most two
- Hence, we obtain the necessary and sufficient condition:

$$\frac{n}{2^r} \leq 2$$

Worst-case Analysis of Binary Search

Worst-case Analysis

- Without recursive calls, we spend $O(1)$ time in the function
- Worst-case runtime = $\underbrace{\text{"maximum \# of recursive calls"}}_r \cdot O(1)$
- Observe that in iteration i the size of the array is at most half the size than in iteration $i - 1$
- We stop as soon as the size of the array is at most two
- Hence, we obtain the necessary and sufficient condition:

$$\frac{n}{2^r} \leq 2$$

Solving $\frac{n}{2^r} \leq 2$ yields $r \geq \log n - 1$.

Worst-case Analysis of Binary Search

Worst-case Analysis

- Without recursive calls, we spend $O(1)$ time in the function
- Worst-case runtime = $\underbrace{\text{"maximum \# of recursive calls"}}_r \cdot O(1)$
- Observe that in iteration i the size of the array is at most half the size than in iteration $i - 1$
- We stop as soon as the size of the array is at most two
- Hence, we obtain the necessary and sufficient condition:

$$\frac{n}{2^r} \leq 2$$

Solving $\frac{n}{2^r} \leq 2$ yields $r \geq \log n - 1$. Hence, $r = \lceil \log n - 1 \rceil \leq \log n$ iterations are enough.

Worst-case Analysis of Binary Search

Worst-case Analysis

- Without recursive calls, we spend $O(1)$ time in the function
- Worst-case runtime = $\underbrace{\text{"maximum \# of recursive calls"}}_r \cdot O(1)$
- Observe that in iteration i the size of the array is at most half the size than in iteration $i - 1$
- We stop as soon as the size of the array is at most two
- Hence, we obtain the necessary and sufficient condition:

$$\frac{n}{2^r} \leq 2$$

Solving $\frac{n}{2^r} \leq 2$ yields $r \geq \log n - 1$. Hence, $r = \lceil \log n - 1 \rceil \leq \log n$ iterations are enough.

Worst-case runtime of Binary Search: $O(\log n)$