Runtime of Algorithms

Consider an algorithm $A$ for a specific problem $P$.

**Set of Potential Inputs**

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

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

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

**Average-case Runtime:** $\frac{1}{|S(n)|} \sum_{I \in S(n)} T(I)$
Linear Search:

**Input:** An array $A$ of $n$ integers from the range $\{0, 1, 2, \ldots, k - 1\}$, for some integer $k$, an integer $t \in \{0, 1, 2, \ldots, k - 1\}$

**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 $i \leq n - 2$ and $A[n - 1] = t$

**Best-case Runtime:** $O(1)$
On any input with $A[0] = t$

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

---

```
Require: Array $A$, integer $t$
for $i = 0, \ldots, n - 1$ do
    if $A[i] = t$ then
        return 1
    return 0
```
Possible Inputs of Length $n$

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

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

**Simplification:** Suppose that $k = 2$. Then $|S(n)| = 2^n$

**Average-case Runtime** (suppose that $t = 1$)

$$\text{AVG} = \frac{1}{|S(n)|} \sum_{A \in S(n)} \text{“left-most pos. } i \text{ such that } A[i] = 1\text{“} + 1$$

$$= 2^{-n} \left( \sum_{i=0}^{n-1} |\{A : \text{left-most 1 is at pos. } i\}| \cdot (i + 1) + n \right).$$
Average-case Analysis of Linear Search (continued)

\[
2^{-n} \left( \left( \sum_{i=0}^{n-1} |\{A : \text{left-most 1 is at pos. } i\}| \cdot (i + 1) \right) + n \right)
\]

\[
\begin{array}{ccccccc}
0 & 0 & 0 & 0 & \ldots & 0 & 1 \\
| & | & | & | & | & | & \\
\text{i times} & \text{n–i–1 times}
\end{array}
\]

\[
= 2^{-n} \left( \left( \sum_{i=0}^{n-1} 2^{n-1-i} \cdot (i + 1) \right) + n \right) = \left( \sum_{i=0}^{n-1} \frac{i + 1}{2^{i+1}} \right) + n2^{-n}
\]

\[
\leq O(1) + 1 = O(1)
\]

→ Average-case runtime of linear search with \( k = 2 \) is \( O(1) \)

**Question:** Average-case runtime of linear search for \( k > 2 \)?
How to bound $\sum_{i=0}^{n-1} \frac{i}{2^i}$:

$$S_{n-1} := \sum_{i=0}^{n-1} \frac{i}{2^i}.$$  

**Trick:** Consider $\frac{1}{2} S_{n-1}$

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

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

$$S_{n-1} - \frac{1}{2} S_{n-1} = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots + \frac{1}{2^{n-1}} - \frac{n-1}{2^n}$$

$$= \left( \sum_{i=1}^{n-1} \frac{1}{2^i} \right) - \frac{n-1}{2^n} = \frac{\frac{1}{2^n} - \frac{1}{2}}{\frac{1}{2} - 1} - \frac{n-1}{2^n} = O(1).$$
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 $t$

```plaintext
if $|A| \leq 2$ then
    Check $A[0]$ and $A[1]$ and return answer
else if $A[\lfloor n/2 \rfloor] = t$ then
    return $\lfloor n/2 \rfloor$
else if $A[\lfloor n/2 \rfloor] > t$ then
    return Binary-Search($A[0, \ldots, \lfloor n/2 \rfloor - 1]$)
else
    return $\lfloor n/2 \rfloor + 1 + Binary-Search(A[\lfloor n/2 \rfloor + 1, n - 1])$
```

Algorithm Binary-Search
Worst-case Analysis of Binary Search

**Worst-case Analysis**

- Without the recursive calls, we spend $O(1)$ time in the function.
- **Worst-case runtime** = 
  
  \[
  \text{"maximum number of recursive calls"} \cdot O(1)
  \]
  
  \[
  r
  \]
- Observe that in iteration $i$ the size of the array is at 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:

\[
\frac{n}{2r} \leq 2
\]

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

**Worst-case runtime of Binary Search**: $O(\log n)$
Proofs by Induction and Loop Invariants
Proofs by Induction

- Correctness of an algorithm often requires proving that a property holds throughout the algorithm (e.g. loop invariant)
- This is often done by induction
- We will first discuss the “proof by induction” principle
- We will use proofs by induction for proving loop invariants (soon) and for solving recurrences (later)
**Geometric Series**: Let $n$ be an integer and let $x \neq 1$. Then:

$$
\sum_{i=0}^{n} x^i = \frac{x^{n+1} - 1}{x - 1}.
$$

**Proof.** (by induction on $n$)

- **Base case.** ($n = 0$)

  $$
  \sum_{i=0}^{0} x^i = x^0 = 1 \quad \text{and} \quad \frac{x^{n+1} - 1}{x - 1} = \frac{x - 1}{x - 1} = 1. \ 
  \checkmark
  $$

- **Induction Step.** Suppose the formula holds for $n$. We will prove that it also holds for $n + 1$:

  $$
  \sum_{i=0}^{n+1} x^i = x^{n+1} + \sum_{i=0}^{n} x^i = x^{n+1} + \frac{x^{n+1} - 1}{x - 1}
  $$

  $$
  = \frac{x^{n+1}(x - 1) + x^{n+1} - 1}{x - 1} = \frac{x^{n+2} - 1}{x - 1}. \ 
  \checkmark
  $$
Structure of a Proof by Induction

- **Statement to prove:** For example, for all \( n \geq k \) \( P(n) \) is true

\[
\forall n \geq 0 : \sum_{i=0}^{n} i = \frac{n(n+1)}{2}.
\]

- **Base case:** Prove that \( P(k) \) holds

\[
n = 0 : \sum_{i=0}^{0} i = 0 = \frac{0 \cdot (0 + 1)}{2}. \checkmark
\]

- **Induction hypothesis:** Assume that \( P \) holds for some \( n \)
  (Strong induction: for all \( m \) with \( k \leq m \leq n \))

- **Induction step:** Prove that \( P(n + 1) \) holds

\[
\sum_{i=0}^{n+1} i = n + 1 + \sum_{i=0}^{n} i = n + 1 + \frac{n(n+1)}{2} = \frac{(n+1)(n+2)}{2}. \checkmark
\]
Induction without sums

**Exercise** Prove that \(n^3 - n\) is divisible by 3, for \(n \geq 2\)

**Proof.**

- **Base case.** \((n = 2)\) \(2^3 - 2 = 6\), which is divisible by 3 ✓
- **Induction step.** Assume statement holds for \(n\). Then:

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

By the induction hypothesis \(n^3 - n\) is divisible by 3. The term \(3(n^2 + n)\) is clearly divisible by 3. The sum of two numbers that are divisible by 3 is also divisible by 3. □
Exercise Prove that $n^3 - n$ is divisible by 3, for $n \geq 2$

Proof.

$$n^3 - n = n(n^2 - 1) = n(n + 1)(n - 1).$$

Observe that $n - 1$, $n$, $n + 1$ are three consecutive numbers larger equal to 1 (for $n \geq 2$). Hence, one of them is necessarily divisible by 3.