Quiz

Let $f,g:\mathbb{N}\to\mathbb{R}^+$ be some functions. Then, we have $\max\{f(n),g(n)\}\le O(f(n)+g(n))$.

True

False

$$\max\{f(n),g(n)\}\le f(n)+g(n)\le O(f(n)+g(n))$$

$n^n \ge \Omega(100^n)$

True

False

$$\lim_{n\to\infty}\frac{n^n}{100^n}=\lim_{n\to\infty}\left(\frac{n}{100}\right)^n=\lim_{n\to\infty}e^{\ln\left(\left(\frac{n}{100}\right)^n\right)}=\lim_{n\to\infty}e^{n\cdot\ln\left(\frac{n}{100}\right)}$$

$$=\lim_{n\to\infty}e^{n\,(\ln(n)-100)}$$

$$=\infty$$

$n^5+10n^4\log(n)=\Theta(n^5\log(n))$.

True

False

$2^{\log_{10}(n)}=\Theta(2^{\ln(n)})$.

True

False

$$\lim_{n\to\infty}\frac{2^{\log_{10}(n)}}{2^{\ln(n)}}=\lim_{n\to\infty}2^{\log_{10}(n)-\ln(n)}=\lim_{n\to\infty}2^{\frac{\ln(n)}{\ln(10)}-\ln(n)}$$

$$=\lim_{n\to\infty}2^{\ln(n)\cdot\underbrace{\left(\underbrace{\frac{1}{\ln(10)}}_{<1}-1\right)}_{<0}}$$

$$=0$$

Suppose you have some algorithm A and let an increasing function $T(n)$ be its runtime on an input of size n. Assume you know that $\underbrace{T(n)\le T(n/2)+C}_{\text{binary search}}$ for some constant $C>0$ and that $T(1)=1$. Then, $T(n)\le O(\log n)$.

binary search

True

False

Suppose you have some algorithm A and let an increasing function $T(n)$ be its runtime on an input of size n. Assume you know that $\underbrace{T(n) \geq 2 \cdot T(n/2) + dn}_{}$ for some constant $d > 0$ and that $T(1) = 10$. Then, $T(n) \geq \Omega(n^{1.5})$.

Merge Sort $\leq O(n \log n) \not\geq \Omega(n^{1.5})$

True

False

Linear search has a worst-case complexity of $\Theta(n)$ on sorted arrays.

True

False

Alice has designed a new comparison-based search algorithm which takes as input a sorted array of n integers along with a target value x and works by iteratively squaring the search range. The algorithm is correct. She claims that this algorithm runs in time $O(\log(\log(n)))$ in the worst case.

Untere Schranke $\Omega(\log n)$

Select one:

a. Alice is mistaken, her algorithm is slower than $O(\log(\log(n)))$.

b. Yes, iterated squaring leads to a $O(\log(\log(n)))$ runtime.

c. Without knowing the details of the algorithm, we cannot make a statement about its worst-case runtime.

Which sorting algorithm from the lecture does the pseudocode snippet below implement?

for $j = 1, 2, \ldots, n-1$ **do:**
  $max = 1$
  for $k = 1, 2, \ldots, n-j+1$ **do:**
    if $A[k] > A[max]$ **then:**
      $max = k$
  if $max < n-j+1$ **then:**
    Swap $A[max]$ and $A[n-j+1]$

findet Index des Maximum

a. Selection Sort
b. Insertion Sort
c. Merge Sort
d. Bubble Sort

Consider the following pseudocode snippet:

$i \leftarrow 1$
while $i \leq n$:
  $j \leftarrow 1$
  while $j \leq i$:
    $f()$
    $j \leftarrow j+1$
  $i \leftarrow i+1$

Which of the following expressions correctly describes the exact number of calls to f?

a. $\sum_{i=1}^{n}\left(\sum_{j=1}^{i}1\right)$
b. $\sum_{i=1}^{n} i \cdot \left(\sum_{j=1}^{n}1\right)$
c. $\sum_{i=1}^{n}\left(1+\sum_{j=1}^{i}1\right)$
d. $\sum_{i=1}^{n}\left(\sum_{j=1}^{n}1\right)$

Logarithmusgesetze

Bezeichnung	Formel	Beispiel	Erklärung
Basiswechsel	$\log_a(x) = \frac{\log_b(x)}{\log_b(a)}$	$\log_2(8) = \frac{\log_{10}(8)}{\log_{10}(2)}$	Ermöglicht den Wechsel der Basis
Exponentenregel	$\log_a(x^r) = r \cdot \log_a(x)$	$\log_2(8^3) = 3 \cdot \log_2(8)$	Der Exponent kann vor den Logarithmus gezogen werden
Produktregel	$\log_a(x \cdot y) = \log_a(x) + \log_a(y)$	$\log(2 \cdot 5) = \log(2) + \log(5)$	Produkt wird zu einer Summe
Quotientenregel	$\log_a\left(\frac{x}{y}\right) = \log_a(x) - \log_a(y)$	$\log(10/2) = \log(10) - \log(2)$	Quotient wird zu einer Differenz

Potenzgesetze

Bezeichnung	Formel	Beispiel	Erklärung
Produktregel (gleiche Basis)	$a^m \cdot a^n = a^{m+n}$	$2^3 \cdot 2^4 = 2^7$	Exponenten werden addiert
Quotientenregel (gleiche Basis)	$\frac{a^m}{a^n} = a^{m-n}$	$\frac{3^5}{3^2} = 3^3$	Exponenten werden subtrahiert
Potenz einer Potenz	$(a^m)^n = a^{m \cdot n}$	$(2^3)^4 = 2^{12}$	Exponenten werden multipliziert
Potenz eines Produkts	$(a \cdot b)^n = a^n \cdot b^n$	$(2 \cdot 3)^2 = 2^2 \cdot 3^2$	Exponent verteilt sich auf Faktoren
Potenz eines Quotienten	$\left(\frac{a}{b}\right)^n = \frac{a^n}{b^n}$	$\left(\frac{2}{3}\right)^2 = \frac{2^2}{3^2}$	Exponent verteilt sich auf Zähler und Nenner
Negative Exponenten	$a^{-n} = \frac{1}{a^n}$	$2^{-3} = \frac{1}{8}$	Negativer Exponent bedeutet Kehrwert
Wurzelgesetz	$a^{1/k} = \sqrt[k]{a}$	$8^{1/3} = \sqrt[3]{8} = 2$	Exponent $1/k$ entspricht der k-ten Wurzel

Ableitungsregeln

Regel	Formel	Beispiel
Konstante	$(c)' = 0$	$(5)' = 0$
Potenzregel	$(x^n)' = n \cdot x^{n-1}$	$(x^3)' = 3x^2$
Summenregel	$(f+g)' = f' + g'$	$(x^2 + x)' = 2x + 1$
(Produktregel)	$(f \cdot g)' = f'g + fg'$	$(x \cdot e^x)' = 1 \cdot e^x + x \cdot e^x$
(Quotientenregel)	$\left(\frac{f}{g}\right)' = \frac{f'g - fg'}{g^2}$	$\left(\frac{x}{e^x}\right)' = \frac{1 \cdot e^x - xe^x}{(e^x)^2}$
Kettenregel	$(f(g(x)))' = f'(g(x)) \cdot g'(x)$	$(\sin(2x))' = \cos(2x) \cdot 2$
Exponentialfunktion	$(e^x)' = e^x$	$(e^{3x})' = 3e^{3x}$
Logarithmus	$(\ln(x))' = \frac{1}{x}$	$(\ln(5x))' = \frac{1}{x}$

Exercise 3.1 *Asymptotic growth* **(2 points).** 3.1a für Peergrading

(a) Prove or disprove the following statements. Justify your answer.

(1) For all $a, b > 1$, suppose $n \in \mathbb{N}$ and $n > \max\{a, b\}$, then $\log_a(n) = \Theta(\log_b(n))$.

Let $a, b \in \mathbb{N}$ be arbitrary with $a, b > 1$

$$\log_a(n) = \frac{\log_b(n)}{\log_b(a)} = \frac{1}{\log_b(a)} \cdot \log_b(n) = \Theta(\log_b(n))$$

$\Rightarrow$ All logarithms have the same asymptotic growth!

(The base doesn't matter)

(2) For all $a, b \geq 1$, suppose $n \in \mathbb{N}$, then $a^n = \Theta(b^n)$

Counterexample: $a = 1$, $b = 2$

$$\lim_{n\to\infty} \frac{a^n}{b^n} = \lim_{n\to\infty} \frac{1^n}{2^n} = \lim_{n\to\infty} \frac{1}{2^n} = 0$$

$$\overset{\text{Theorem 1.1}}{\Longrightarrow} \quad a^n \neq \Theta(b^n)$$

(2) Prove that $\lim_{n\to\infty} n(e^{1/n} - 1) = 1$.

You may use the following variant of L'Hôpital's rule:

Theorem 2 (L'Hôpital's rule (going to 0)). *Assume that functions $f : \mathbb{R}^+ \to \mathbb{R}$ and $g : \mathbb{R}^+ \to \mathbb{R}$ are differentiable, $\lim_{x\to\infty} f(x) = \lim_{x\to\infty} g(x) = 0$ and for all $x \in \mathbb{R}^+$, $g'(x) \neq 0$. If $\lim_{x\to\infty} \frac{f'(x)}{g'(x)} = C \in \mathbb{R}$ or $\lim_{x\to\infty} \frac{f'(x)}{g'(x)} = \infty$, then*

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = \lim_{x\to\infty} \frac{f'(x)}{g'(x)}.$$

$$\lim_{n\to\infty} n\cdot(e^{\frac{1}{n}} - 1) = \lim_{n\to\infty} \frac{e^{\frac{1}{n}} - 1}{\frac{1}{n}}$$

$\frac{1}{n} \to 0$, $e^{\frac{1}{n}} \to 1$, $e^{\frac{1}{n}} - 1 \to 0$

$$\overset{\text{L'Hôp.}}{=} \lim_{n\to\infty} \frac{-e^{\frac{1}{n}} \cdot \frac{1}{n^2}}{-\frac{1}{n^2}}$$

$\left(\frac{1}{n}\right)' = \left(n^{-1}\right)' = -n^{-2} = -\frac{1}{n^2}$

$$= \lim_{n\to\infty} e^{\frac{1}{n}}$$

weil $\lim_{n\to\infty} e^{f(n)} = e^{\lim_{n\to\infty} f(n)}$

$$= 1$$

Einfacher Lösungsweg mit Substitution $y = \frac{1}{n}$

$$\lim_{n\to\infty} n\cdot(e^{\frac{1}{n}} - 1) = \lim_{y\to 0} \frac{1}{y}\cdot(e^{y} - 1) = \lim_{y\to 0} \frac{e^{y} - 1}{y} \overset{\text{L'Hôp}}{=} \lim_{y\to 0} \frac{e^{y}}{1} = 1$$

(3) Prove that for $n \geq 3$

$$\frac{n^{1/n}-1}{n} = \Theta\left(\frac{\ln(n)}{n^2}\right).$$

You may use results from the earlier exercises.

You may use the following fact:

$\ast$ Let $f, g : \mathbb{R}^+ \to \mathbb{R}$. Suppose that $\lim_{n\to\infty} g(n) = \infty$ and there exists some constant $C \in \mathbb{R}$ such that $\lim_{n\to\infty} f(n) = C$. Then $\lim_{n\to\infty} f(g(n)) = C$.

$$\lim_{n\to\infty} \frac{\frac{n^{\frac{1}{n}}-1}{n}}{\frac{\ln(n)}{n^2}} = \lim_{n\to\infty} \frac{n^{\frac{1}{n}}-1}{\cancel{n}} \cdot \frac{n^{\cancel{2}}}{\ln(n)}$$

$$= \lim_{n\to\infty} \frac{n\,(n^{\frac{1}{n}}-1)}{\ln(n)}$$

$$= \lim_{n\to\infty} \frac{n\cdot(e^{\ln(n^{\frac{1}{n}})}-1)}{\ln(n)}$$

$$= \lim_{n\to\infty} \frac{n\cdot(e^{\frac{\ln(n)}{n}}-1)}{\ln(n)}$$

Sei $f(n) := n\,(e^{\frac{1}{n}}-1)$,
$g(n) := \frac{n}{\ln(n)}$

$$= \lim_{n\to\infty} f(g(n))$$

$= 1$ nach $(\ast)$ weil $\lim_{n\to\infty} g(n) = \lim_{n\to\infty} \frac{n}{\ln(n)} = \infty$

Exercise 3.3 *Counting function calls in loops* **(1 point)**.

For each of the following code snippets, compute the number of calls to f as a function of $n \in \mathbb{N}$. Provide **both** the exact number of calls and a maximally simplified asymptotic bound in Θ notation.

Algorithm 1

(a)
```
i ← 0
while i ≤ n do
    f()
    f()
    i ← i + 1
j ← 0
while j ≤ 2n do
    f()
    j ← j + 1
```

$$\#\text{calls} = \sum_{i=0}^{n} 2 + \sum_{j=0}^{2n} 1$$

$$= (n+1)\cdot 2 + (2n+1)\cdot 1$$

$$= 2n+2+2n+1$$

$$= 4n+3$$

$$= \Theta(n)$$

Algorithm 2

(b)
```
i ← 1
while i ≤ n do
    j ← 1
    while j ≤ i^3 do
        f()
        j ← j + 1
    i ← i + 1
```

$$\#\text{calls} = \sum_{i=1}^{n} \sum_{j=1}^{i^3} 1$$

$$= \sum_{i=1}^{n} i^3$$

$$= \left(\frac{n(n+1)}{2}\right)^2$$

$$= \frac{n^2(n+1)^2}{4}$$

$$= \Theta(n^4)$$

$\sum_{i=1}^{n} i$	$\frac{n(n+1)}{2}$
$\sum_{i=1}^{n} i^2$	$\frac{n(n+1)(2n+1)}{6}$
$\sum_{i=1}^{n} i^3$	$\left(\frac{n(n+1)}{2}\right)^2$

Neue Gruppen

Member 1	Member 2	Member 3
Samuel Burkhardt (sburkhard@student.ethz.ch)	Gaeacrist Salmo (gsalmo@student.ethz.ch)	
Noemi Blanc (nblanc@student.ethz.ch)	Benjamin Stupf (bstupf@student.ethz.ch)	
Nicolas Holzmann (nholzmann@student.ethz.ch)	Timo Bruhin (tbruhin@student.ethz.ch)	
Matthias Lüthi (luethm@student.ethz.ch)	Danijel Jovanovic (danijelj@student.ethz.ch)	
Jonas Lüthi (joluethi@student.ethz.ch)	Patrick Back (pback@student.ethz.ch)	
Dominic Aschmann (daschmann@student.ethz.ch)	Tobias Hagen (tohagen@student.ethz.ch)	
Roman Huber (romahuber@student.ethz.ch)	Mario Malis (mmalis@student.ethz.ch)	
Moritz Becker (mobecker@student.ethz.ch)	Jolanda Bellert (jbellert@student.ethz.ch)	
Yannis Grimm (ygrimm@student.ethz.ch)	Max Landschoof (mlandschoof@student.ethz.ch)	
Mattia Niggli (nigglima@student.ethz.ch)	Siqi Wang (siqwang@student.ethz.ch)	
Haochen Zhu (haoczhu@student.ethz.ch)	Marc Leonard Overbeck (moverbeck@student.ethz.ch)	
Lucien Gees (lugees@student.ethz.ch)	Luca Schnellmann (lschnellman@student.ethz.ch)	Francisco Zanatta Janzen (fzanatta@student.ethz.ch)

Such - Algorithmen

```
LINEAR-SEARCH(A[1..n], b)

for i ← 1, 2, ..., n do
    if A[i] = b then return i
return "nicht gefunden"
```

- für beliebige arrays (auch unsortierte)
- Worst Case Laufzeit $\Omega(n)$

```
BINARY-SEARCH(A[1..n], b)

ℓ ← 1; r ← n                                ▷ Initialer Suchbereich
while ℓ ≤ r do
    m ← ⌊(ℓ + r)/2⌋
    if A[m] = b then return m               ▷ Element gefunden
    else if A[m] > b then r ← m − 1         ▷ Suche links weiter
    else ℓ ← m + 1                          ▷ Suche rechts weiter
return "Nicht vorhanden"
```

- Nur für sortierte arrays

Bsp

BinarySearch(A,31):

$\ell=1$, $m=5$, $r=10$

3 4 7 8 9 20 31 41 59 265

$\ell=6$, $m=8$, $r=10$

~~3 4 7 8 9~~ 20 31 41 59 265

$\ell=6$, $r=7$

~~3 4 7 8 9~~ 20 31 ~~41 59 265~~

$\ell=m=6$, $r=7$

~~3 4 7 8 9~~ 20 31 ~~41 59 265~~

$\ell=r=m=7$

~~3 4 7 8 9~~ ~~20~~ 31 ~~41 59 265~~

- In jeder Iteration halbieren wir den Suchbereich
- Worst Case Laufzeit $\Omega(\log n)$

Sortier - Algorithmen

Bubble-Sort($A[1..n]$)

```
for j ← 1, 2, ..., n do
    for i ← 1, 2, ..., n − 1 do
        if A[i] > A[i + 1] then
            tausche A[i] und A[i + 1]
```

Vergleiche: $n \ (n-1) = \Theta(n^2)$

Vertauschungen: $O(n^2)$

Bsp: $j=3$ $\quad 3 \ 2 \ 1 \ \boxed{6 \ 8 \ 9}$

Inv(3)

Laufzeit: $\Theta(n^2)$

Inv(j) = Nach j Iterationen der äusseren Schleife sind die j grössten Elemente am richtigen Ort

Korrektheit: man zeigt Inv(0) als base case

Inv$(j) \Rightarrow$ Inv$(j+1)$ als Induktionsschritt für $0 \leq j \leq n-1$

Inv$(n) \Rightarrow$ Array ist sortiert

Selection-Sort($A[1..n]$)

```
for j ← n, n − 1, ..., 1 do
    k ← Index des Maximums in A[1, ..., j]
    tausche A[k] und A[j]
```

Vergleiche: $\Theta(n^2)$

Bsp: $j=3$ $\quad 3 \ 2 \ 1 \ \boxed{6 \ 8 \ 9}$

Inv(3)

Vertauschungen: $O(n)$

Laufzeit: $\Theta(n^2)$

Inv(j) = Nach j Iterationen der äusseren Schleife sind die j grössten Elemente am richtigen Ort

---

INSERTION-SORT($A[1..n]$)

```
for j ← 2, 3, ..., n do
    k ← kleinster Index in {1, ..., j − 1} mit A[j] ≤ A[k]   → mit Binary Search
                                                   ▷ A[j] gehört an diese Stelle k
    x ← A[j]                                       ▷ merke A[j]
    verschiebe A[k, ..., j − 1] nach A[k + 1, ..., j]
    A[k] ← x
```

Vergleiche: $O(n \log n)$

Vertauschungen: $O(n^2)$

Laufzeit: $O(n^2)$

Bsp: $j=3$

Inv(3) $\quad k=2$

$\boxed{2 \; 4 \; 9} \; 3 \; 1 \; 8$

Inv(4)

$\boxed{2 \; 3 \; 4 \; 9} \; 1 \; 8$

Inv(j) = Nach j Iterationen der äusseren Schleife ist das Teilarray A[1...j] sortiert (es enthält aber nicht zwangsläufig die j kleinsten Elemente des Arrays).

```
MERGESORT(A[1..n], l, r)                         ▷ sortiert A[l,...,r]

if l < r then
    m ← ⌊(l + r)/2⌋
    MERGESORT(A, l, m)                           ▷ sortiere linke Hälfte
    MERGESORT(A, m + 1, r)                       ▷ sortiere rechte Hälfte
    MERGE(A, l, m, r)                            ▷ verschmelze beide Hälften
```

divide and conquer

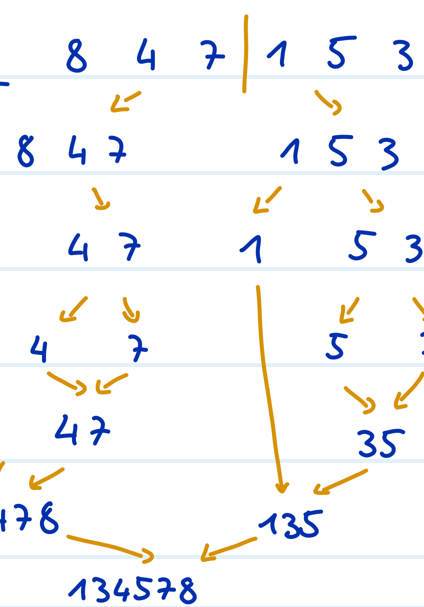
```
MERGE(A[1..n], l, m, r)

B ← neues Array mit r − l + 1 Zellen             ▷ so gross wie A[l,...,r]
i ← l                                            ▷ erstes unbenutztes Element in linker Hälfte
j ← m + 1                                        ▷ erstes unbenutztes Element in rechter Hälfte
k ← 1                                            ▷ nächste Position in B
while i ≤ m and j ≤ r do                         ▷ beide Hälften noch nicht ausgeschöpft
    if A[i] < A[j] then
        B[k] ← A[i]
        i ← i + 1
        k ← k + 1
    else
        B[k] ← A[j]
        j ← j + 1
        k ← k + 1
übernimm Rest links bzw. rechts                  ▷ wenn die andere Hälfte ausgeschöpft ist
kopiere B nach A[l,...,r]
```

Laufzeit von MERGE: $O(n)$

Totale Laufzeit: $T(n) = \underbrace{2 \cdot T\left(\frac{n}{2}\right)}_{\text{zwei Hälften rekursiv sortieren}} + \underbrace{c \cdot n}_{\text{Merge}}$

Zusätzlicher Speicher: $O(n)$ für das Array B

Bsp



g) *Sorting algorithms:*

i) Consider the sequence $6, 5, 4, 1, 2, 3$. How many swaps does `Bubble Sort` perform to sort this sequence? *Give the exact number of swaps required.*

$5 + 4 + 3 = 12$

ii) Consider the sequence $6, 5, 4, 1, 2, 3$. How many swaps does `Selection Sort` perform to sort this sequence? *Give the exact number of swaps required.*

4

iii) Let $n \in \mathbb{N}$ be an even number and consider the sequence with the following structure:

$$2, 1, 4, 3, 6, 5, \ldots, n, n-1.$$

How many swaps does `Insertion Sort` perform to sort this sequence? *Give the exact number, not just the asymptotics.*

$\frac{n}{2}$

- [x] All four algorithms can sort any list of comparable elements correctly.
- [x] Only Merge Sort can guarantee a worst-case time complexity better than $O(n^2)$.
- [ ] Bubble Sort compares distant elements in the array during each pass.
- [ ] Insertion Sort inserts each new element into the already sorted part at the final correct position.
- [ ] Selection Sort may perform fewer comparisons if the input list is already sorted.
- [ ] Merge Sort always splits the list into two equal halves.
- [ ] Bubble Sort and Insertion Sort both have an average runtime of $O(n \log n)$.
- [x] Merge Sort's runtime does not depend on the initial order of the elements.
- [x] Selection Sort performs fewer swaps than Bubble Sort in most cases.
- [ ] Insertion Sort requires extra arrays to perform its sorting.
- [x] Merge Sort typically requires additional memory proportional to the size of the input.
- [ ] Selection Sort always performs exactly n–1 comparisons.
- [ ] Bubble Sort can terminate early when no swaps occur during a pass.

✓ Merge Sort is an example of a “divide and conquer” algorithm.

✗ Bubble Sort is usually faster than Merge Sort on large, random datasets.

✗ Selection Sort adapts its runtime when the input is already sorted. $\Theta(n^2)$ Vergleiche

✓ Merge Sort can be implemented recursively or iteratively.

✓ Bubble Sort, Insertion Sort, and Selection Sort all sort in-place.

✗ Merge Sort sorts in-place without using additional arrays.

✓ All four algorithms determine order by comparing pairs of elements.

✓ Merge Sort divides the list until each sublist contains exactly one element.

✓ The worst-case runtime of Bubble Sort and Selection Sort is the same. both $O(n^2)$

✓ Insertion Sort moves elements to make space for the current value being inserted.

✗ Merge Sort avoids element comparisons when merging sublists.

✗ Selection Sort and Bubble Sort perform the same number of swaps in total.

✓ Insertion Sort can outperform Merge Sort on very small arrays.

~~Merge Sort can be used on linked lists as well as arrays~~.

✓ Bubble Sort is primarily used in practical applications for large datasets.

✗ Any sorting algorithm is in $\Omega(n \log n)$. Bucket Sort

✓ Selection Sort can be better than Bubble Sort.

✗ Merge Sort is always faster than Insertion Sort.

✓ Merge Sort has $O(n^2)$ comparisons. $O(n \log n) \leq O(n^2)$ This is important!!!

✗ Insertion Sort runs in $O(n)$ time on sorted input. binary search

✗ Bubble Sort’s best case is faster than Selection Sort’s best case. (depends on exact implementation)

✗ Merge Sort’s runtime depends on how sorted the array already is.

✗ Selection Sort’s number of comparisons depends on input order.

✗ Every comparison-based sort must perform $\Omega(n \log n)$ comparisons in every case.
worst