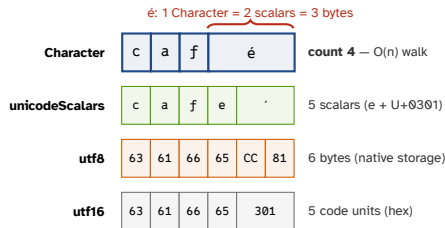


In one line: A Swift String is a **UTF-8 buffer** (since Swift 5) seen as a **bidirectional** collection of Characters, where a Character is an **extended grapheme cluster** — one user-perceived character of variable byte width. Hence no `s[3]`, count is **$O(n)$** , and slices (Substring, ArraySlice) **share and pin** their parent's storage. All stdlib collections are value types with **copy-on-write**.

Picture — one string, four views



"café\u{301}" == "caf\u{E9}" is **true** (canonical equivalence) though the bytes differ. Family emoji: 1 Character, 7 scalars, 25 UTF-8 bytes, 11 UTF-16 units.

Strings — how it works

- Native storage is **UTF-8**; strings ≤ 15 UTF-8 bytes live **inline** (small-string, no heap) on 64-bit.
- `String.Index` is an opaque byte position. `index(_:offsetBy:)` is $O(n)$. An index is valid **only for the string it came from** and **until it mutates**; never reuse it across strings or edits.
- `isEmpty` is $O(1)$, `count` $O(n)$ — never `count == 0`. `s.utf8.count` is $O(1)$ for native strings.
- Substring (from split, prefix, ranges) **shares the whole base buffer**: `String(sub)` before storing. Take some `StringProtocol` to accept both.
- `==`, `hasPrefix`, hashing use canonical equivalence; `<` is *not* locale-aware — UI sort: `localizedStandardCompare`.

Example

```
let s = "café\u{301}" // "café", decomposed
s.count; s.unicodeScalars.count; s.utf8.count // 4, 5, 6
let i = s.index(s.startIndex, offsetBy: 3) // s[3] won't compile
s[i] // "é"
let first = log.split(separator: "\n")[0] // Substring
cache.append(String(first)) // copy out
var a = [Int](); a.reserveCapacity(n) // ONE allocation
let hit = big.lazy.map(f).first(where: p) // no temp array
var q = Deque<Job>(); q.prepend(job) // O(1) front
var seen: OrderedSet<ID> = []; seen.append(id) // unique, ordered
var pq = Heap<Int>(); pq.insert(5); pq.popMin() // O(log n)
```

Picture — a slice pins its parent



Collections — how it works

- Array:** one contiguous heap buffer; append **amortised $O(1)$** (capacity grows geometrically, $\times 2$); `reserveCapacity` = one allocation. Mutation copies only if the buffer is shared (COW) — capacity $\neq$ uniqueness.
- Set / Dictionary:** open-addressed hash table, per-process **random seed** (SipHash, anti-HashDoS): average $O(1)$, order unspecified and **different every launch**.
- Hashable contract:** `a == b` $\Rightarrow$ equal hashes. Feed `hash(into:)` exactly the fields `==` uses; never mutate a key while it is inside a Set/Dictionary.
- Slices** (ArraySlice, Substring) keep the parent's **indices** (`a[5..<9].startIndex == 5`) and its **whole buffer** alive.
- `.lazy` fuses `map/filter` with no temp arrays, but **re-runs** the closures on every iteration.

Complexity — pick the structure

	[i]	end +	front +	contains	order
Array	$O(1)$	$O(1)^*$	$O(n)$	$O(n)$	insertion
Set, Dictionary	—	$O(1)^*$	—	$O(1)$ avg	none
Deque	$O(1)$	$O(1)^*$	$O(1)^*$	$O(n)$	insertion
OrderedSet/Dict	$O(1)$	$O(1)^*$	$O(n)$	$O(1)$	insertion
Heap (min-max)	—	$O(\log n)$	—	$O(n)$	min/max $O(1)$
String	$O(n)$	$O(1)^*$	$O(n)$	$O(n)$	—

* amortised. Removing from the middle is $O(n)$ in every array-backed one (incl. `OrderedSet`). swift-collections: import Collections (or DequeModule, OrderedCollections, HeapModule); also BitSet, TreeSet/TreeDictionary (persistent).

Interview traps

- !** `split` returns [Substring] — store them and you keep the whole input alive.
- !** Testing or persisting Set/Dictionary order — it changes per launch.
- !** `removeFirst()` on Array in a queue loop = $O(n^2)$ $\rightarrow$ Deque.
- !** Iterating a slice with `0..<slice.count` — use `slice.indices`.
- !** `reserveCapacity` inside the append loop defeats geometric growth.

Remember

"Graphemes cost a walk; slices hold the whole; hashes forget order." Both ends $\rightarrow$ Deque · unique+ordered $\rightarrow$ OrderedSet · priority $\rightarrow$ Heap.

Likely questions

- Why no `str[5]`? — Characters are variable-width; Int offset is $O(n)$, so `String.Index`.
- Family emoji (4 people + 3 ZWJ) .count? — 1 (one grapheme cluster).
- Array vs ContiguousArray? — the latter never bridges to NSArray.
- LRU cache? — Dictionary + doubly-linked list for $O(1)$; OrderedDictionary is $O(n)$ per move.