$\boxed{\text{Sorting}}$

Stable

Time

Inplace → No additional memory necessary for storing the output of sorting

$k_2 = k_3$

$(k_1, v_1), (k_2, v_2), (k_3, v_3) \ldots (k_n, v_n)$

Key

$(k_1, v_1), (k_3, v_3), (k_2, v_2) \ldots (k_n, v_n)$

BST, Splay Tree

1. Insertion sort

S → Sorted portion

I → Input

Start with an empty list S and unsorted list I of n-items.

$\Theta(n)$ → for (each item x in I) {

$\boxed{\text{Insert}}$ x into S in sorted order. → $\Theta(n)$

}

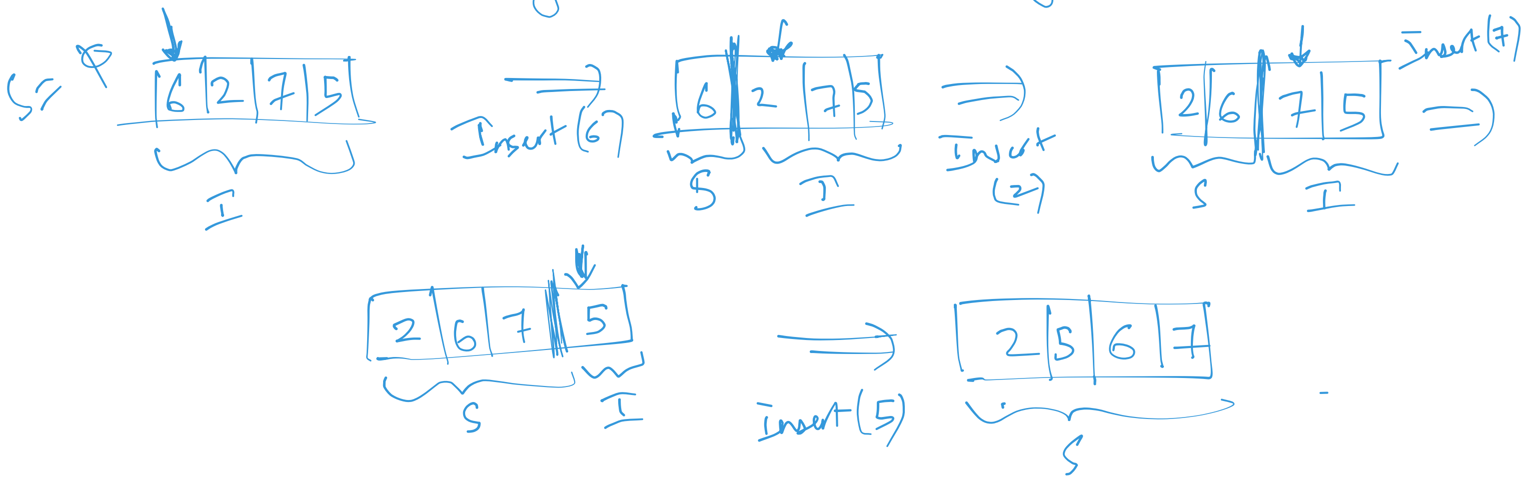
If S is a linked list, $\Theta(n)$ time to find the right position.

If S is an array, $\Theta(n)$ time to shift the higher items over.

Insertion sort : $\Theta(n^2)$

Inplace sort if S is an array

↳ Take all the items and sort without any additional memory.



* Already almost sorted ⇒ Runs much faster.
  ⇒ proportional to the number of swaps.

* Finding the right compromise between linked list and arrays ⇒

  If S is a balanced BST, running time is $\Theta(n \lg n)$

---

② Selection Sort : $\Theta(n^2)$ always.

↳ Worse than Insertion sort

Start with an empty list S and unsorted list I of n items.

```
for (i = 0;  i < n ;  ++i) {
    x → item in I with the smallest key.      ← Search
    Remove x from I         ← Selection
    Append x to the end of S.
}
```

Whether S is an array or a linked list,
Searching for min item in an unsorted list
takes $\Theta(n)$ time.

$\therefore$ $\Theta(n^2)$ even in the best case. (almost sorted input)

Eg. - $\underbrace{\boxed{6 \mid 2 \mid 7 \mid 5}}_{I}$ $\xrightarrow[\text{min in } I]{\text{Search for}}$ $\underbrace{\boxed{2}}_{S}\underbrace{\boxed{6 \mid 7 \mid 5}}_{I}$ $\xrightarrow[\text{min in } I]{\text{Search for}}$

Inplace Algo

$\underbrace{\boxed{2 \mid 5}}_{S}\underbrace{\boxed{6 \mid 7}}_{I}$ $\xrightarrow[\text{min in } I]{\text{Search for}}$ $\underbrace{\boxed{2 \mid 5 \mid 6}}_{S}\underbrace{\boxed{7}}_{I}$ $\Longrightarrow$ $\underbrace{\boxed{2 \mid 5 \mid 6 \mid 7}}_{S}$

* Replace I with a heap $\Longrightarrow$ $\boxed{\text{Heap Sort}}$

---

(3) Heap Sort : Selection sort where I is a binary heap.

Start with an empty list S and unsorted list I of n items.

Throw all items in I into a heap h (ignoring the heap-order property)

h.bottomUpHeap(); // $\Theta(n)$ time.

$\Theta(n)$ for $(i = 0;\ i < n;\ ++i)$ {

x = h.removeMin(); // $\Theta(\lg n)$

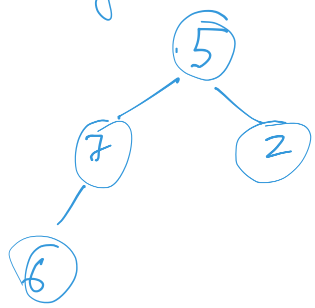
Append x to the end of S

}

$\boxed{\text{Heap sort : } O(n \lg n)}$

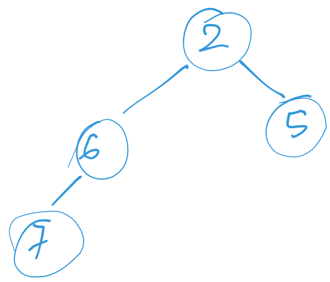
Inplace sort → Maintain Heap at the end of the array, $\boxed{\text{backward}}$

$\underbrace{\boxed{6 \mid 2 \mid 7 \mid 5}}_{T}$



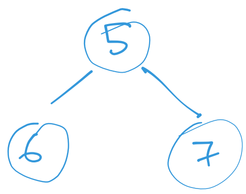
bottom Up Heap ↓

$\underbrace{\boxed{7 \mid 5 \mid 6 \mid 2}}_{T}$



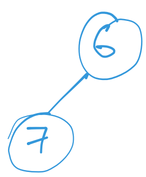
↓ remove Min ( )

$\underbrace{\boxed{2}}_{S} \Vert \underbrace{\boxed{7 \mid 6 \mid 5}}_{T}$



↓ remove Min ( )

$\underbrace{\boxed{2 \mid 5}}_{S} \Vert \underbrace{\boxed{7 \mid 6}}_{T}$



↓ remove Min ( )

$\underbrace{\boxed{2 \mid 5 \mid 6}}_{S} \Vert \underbrace{\boxed{7}}_{T}$

Heap sort : In-place sort

* Excellent for sorting Arrays.
but clumsy for linked list.