Problème 1.

Consider the following sequence of values $S = [3, 42, 39, 86, 49, 89, 99, 20, 88, 51, 64]$

Note

We can represent tree textually via the following representation

```
13 (
        11 (
                8 (
                        2
                        4
                )
                12 (
                        *
                        1
                )
        )
         7 (
                 5
                 6 (
                         99
                         *
                 )
        )
)
```

Where we use $*$ as a placeholder for a missing child for those nodes that only have a single child.

P1.1

Draw the min heap (as a tree) obtained by adding the values in S in sequence. Show each step

1. $S = [3]$. The root of the heap.

```
3
```

2. $S = [3, 42]$. Added to the left of the root.

```
3 (
  42
  *
)
```

3. $S = [3, 42, 39]$. Added to the right of the root.

```
3 (
  42
  39
)
```

4. $S = [3, 42, 39, 86]$. Added to the left of the left child of the root. (42 < 86)

```
3 (
  42 (
    86
    *
  )
  39
)
```

5. $S = [3, 42, 39, 86, 49]$. Added to the right of 42.

```
3 (
  42 (
    86
    49
  )
  39
)
```

6. $S = [3, 42, 39, 86, 49, 89]$. Added to the left of 39.

```
3 (
  42 (
    86
    49
  )
  39 (
    89
  )
)
```

7. $S = [3, 42, 39, 86, 49, 89, 99]$. Added to the right of 39.

```
3 (
  42 (
    86
    49
  )
  39 (
    89
    99
  )
)
```

8. $S = [3, 42, 39, 86, 49, 89, 99, 20]$. 20 becomes left child of 86. (20 < 86) then swap. (20 < 42) then swap.

```
3 (
  20 (
    42
      (
        86
        *
      )
    49
  )
  39 (
    89
    99
  )
)
```

9. $S = [3, 42, 39, 86, 49, 89, 99, 20, 88]$. 88 becomes right of 42

```
3 (
  20 (
    42 (
      86
      88
    )
    49
  )
  39 (
    89
    99
  )
)
```

10. $S = [3, 42, 39, 86, 49, 89, 99, 20, 88, 51]$. 51 becomes right of 49

```
3 (
  20 (
    42 (
      86
      88
    )
    49 (
      51
      *
    )
  )
  39 (
    89
    99
  )
)
```

11. $S = [3, 42, 39, 86, 49, 89, 99, 20, 88, 51, 64]$. 64 becomes right of 49

```
3 (
  20 (
    42 (
      86
      88
    )
    49 (
      51
      64
    )
```

```
  )
  39 (
    89
    99
  )
)
```

P1.2

Draw the max heap (as a tree) obtained by adding the values in S in sequence. Show each step

1. $S = [3]$. The root of the heap.

   ```
   3
   ```

2. $S = [3, 42]$. 42 becomes the root, 3 becomes left child.

```
42 (
  3
  *
)
```

3. $S = [3, 42, 39]$. 39 becomes right child.

```
42 (
  3
  39
)
```

4. $S = [3, 42, 39, 86]$. 86 becomes root. 42 becomes left child, 3 becomes left child of 42.

```
86 (
  42 (
    3
    *
  )
  39
)
```

5. $S = [3, 42, 39, 86, 49]$. 49 becomes left child of 86, swap 42, 42 becomes right child of 49.

```
86 (
  49 (
    3
    42
  )
  39
)
```

6. $S = [3, 42, 39, 86, 49, 89]$. 89 become routes, swap 86, 49.

```
89 (
  49 (
    3
    42
  )
  86 (
    39
    *
  )
)
```

7. $S = [3, 42, 39, 86, 49, 89, 99]$. 99 becomes root, swap 89, 49.

```
99 (
  49 (
    3
    42
  )
  89 (
    39
    86
  )
)
```

8. $S = [3, 42, 39, 86, 49, 89, 99, 20]$. 20 swap with 3, 3 becomes left child of 20.

```
99 (
  49 (
    20 (
```

```
      3
      *
    )
    42
    )
  89 (
    39
    86
  )
)
```

9. $S = [3, 42, 39, 86, 49, 89, 99, 20, 88]$. 88 becomes left child of 99, swap 49, 20.

```
99 (
  88 (
    49 (
      20 (
        3
        *
      )
      *
    )
    42
  )
  89 (
    39
    86
  )
)
```

10. $S = [3, 42, 39, 86, 49, 89, 99, 20, 88, 51]$. 51 becomes right child of 88, swap 42, 20.

```
99 (
  88 (
    51 (
      42 (
        20 (
          3
          *
        )
      )
      *
    )
    49
  )
```

```
    89 (
      39
      86
    )
  )
```

11. $S = [3, 42, 39, 86, 49, 89, 99, 20, 88, 51, 64]$. 64, pushes 49 down.

```
99 (
  88 (
    51 (
      42 (
        20 (
          3
          *
        )
      )
      *
    )
    64 (
      49
      *
    )
  )
  89 (
    39
    86
  )
)
```

P1.3

Draw the binary search tree obtained by adding the values in S in sequence. Show each step

1. $S = [3]$. The root of the tree.

```
3
```

2. $S = [3, 42]$. 42 becomes the right child of 3.

```
3 (
  *
  42
)
```

3. $S = [3, 42, 39]$. 39 becomes the left child of 42.

```
3 (
  *
  42 (
    39
    *
  )
)
```

4. $S = [3, 42, 39, 86]$. 86 becomes the right child of 42.

```
3 (
  *
  42 (
    39
    86
  )
)
```

5. $S = [3, 42, 39, 86, 49]$. 49 becomes the left child of 86.

```
3 (
  *
  42 (
    39
    86 (
      49
      *
    )
  )
)
```

6. $S = [3, 42, 39, 86, 49, 89]$. 89 becomes the right child of 86.

```
3 (
  *
  42 (
    39
    86 (
      49
      89
    )
  )
)
```

7. $S = [3, 42, 39, 86, 49, 89, 99]$. 99 becomes the right child of 89.

```
3 (
  *
  42 (
    39
    86 (
      49
      89 (
        *
        99
      )
    )
  )
)
```

8. $S = [3, 42, 39, 86, 49, 89, 99, 20]$. 20 becomes the left child of 39.

```
3 (
  *
  42 (
    39 (
      20
      *
    )
    86 (
      49
      89 (
        *
        99
      )
    )
```

```
  )
)
```

9. $S = [3, 42, 39, 86, 49, 89, 99, 20, 88]$. 88 becomes the right child of 86.

```
3 (
  *
  42 (
    39 (
      20
      *
    )
    86 (
      49
      89 (
        88
        99
      )
    )
  )
)
```

10. $S = [3, 42, 39, 86, 49, 89, 99, 20, 88, 51]$. 51 becomes the right child of 49.

```
3 (
  *
  42 (
    39 (
      20
      *
    )
    86 (
      49 (
        *
        51
      )
      89 (
        88
        99
      )
    )
  )
)
```

11. $S = [3, 42, 39, 86, 49, 89, 99, 20, 88, 51, 64]$. 64 becomes the left child of 51.

```
3 (
        *
        42 (
                39 (
                        20
                        *
                )
                86 (
                        49 (
                                *
                                51 (
                                        *
                                        64
                                )
                        )
                        89 (
                                88
                                99
                        )
                )
        )
)
```

Problème 2.

Given an ordered list L and value v, the `LowerBound` algorithm provide the position p in list L such that p is the first offset in L of a value larger-equal to v. Hence, $v \leq L[p]$ (or, if no such offset exists, $p = |L|$). The `LowerBound` algorithm does so in $\Theta(\log_2(|L|))$ comparisons. Argue that `LowerBound` is *worst-case optimal*: any algorithm that finds the correct position p for any inputs L and v using only comparisons will require $\Theta(\log_2(|L|))$ comparisons.

Solution

For a list of size $|L|$ there are $|L| + 1$ possible outcomes for the position p in the list. The minimum height of a binary tree needed for $|L| + 1$ outcomes is $\log_2(|L| + 1)$ (at most 2^h leaves or $2^h \geq |L| + 1 \rightarrow h \geq \log_2(|L| + 1)$

From Stirling's approximation, comparison-based sorting algorithm lower bound is $\Omega(n \log(n))$. Given that the algorithm operates in $\Theta(\log_2(|L|))$ comparisons, it matches with the theoretical lower bound for the search algorithm. Therefore, no comparison-based algorithm can guarantee a better worst-case performance for position p, making `LowerBound` the worst-case optimal.

Problème 3.

Min heaps and max heaps allow one to efficiently store values and efficiently look up and remove the *smallest values* and *largest values*, respectively. One cannot easily remove the largest value from a min heap or the smallest value from a max heap, however.

> **P3.1**
>
> Assume a value v is a part of a min heap of at-most n values and that we know v is stored at position p in that heap. Provide an algorithm that can remove v from the heap in worst-case $\mathcal{O}(\log_2(n))$

Algorithm RemoveValue$(heap, p)$

```
procedure RemoveValue(heap, i)
    n ← heap.length
    temp ← heap[p]
    heap[p] ← heap[n]
    heap[n] ← temp
    heap ← heap[: n]
    HeapifyDown(heap, p)
end procedure
```

Algorithm HeapifyDown$(heap, p)$

```
procedure HeapifyDown(heap, i)
    n ← size of heap
    while lchild(i) ≤ n do
        left ← lchild(i)
        right ← rchild(i)
        smallest ← i
        if left ≤ n and heap[left] < heap[smallest] then
            smallest ← left
        end if
        if right ≤ n and heap[right] < heap[smallest] then
            smallest ← right
        end if
        if smallest = i then
            break
        else
            Swap heap[i] with heap[smallest]
            i ← smallest
        end if
    end while
```

end procedure

ⓘ P3.2

Provide a data structure that allows one to efficiently store values and efficiently look up and remove *both* the smallest and the largest values: all three of these operations should be supported in $\Theta(\log_2(n))$

We will implement a Double-ended Priority Queue (DEPQ), which is a min-max heap.

Algorithm Insert$(heap, v)$

---

procedure Insert$(heap, v)$
  $heap.push(v)$
  Swim$(heap, \text{size}(heap))$
end procedure

Algorithm RemoveMin$(heap)$

---

procedure RemoveMin$(heap)$
  $n \leftarrow \text{size}(heap)$
  $temp \leftarrow heap[1]$
  $heap[1] \leftarrow heap[\text{size}(heap)]$
  $heap[n] \leftarrow temp$
  $heap \leftarrow heap[: n]$
  Sink$(heap, 1)$
end procedure

Algorithm RemoveMax$(heap)$

---

procedure RemoveMax$(heap)$
  $maxPos \leftarrow \text{argmax}\{heap[2], heap[3]\}$
  $heap[maxPos] \leftarrow heap[\text{size}(heap)]$
  remove last el from$heap$
  Sink$(heap, maxPos)$
end procedure

Algorithm Swim$(heap, i)$

---

procedure Swim$(heap, i)$
  while $i > 1$ **do**
    $parent \leftarrow \lfloor i/2 \rfloor$
    $grandParent \leftarrow \lfloor parent/2 \rfloor$
    if $(i \mod 2 = 0$ and $heap[i] < heap[parent])$ or $(i \mod 2 \neq 0$ and $heap[i] > heap[parent])$ **then**
      Swap$(heap[i], heap[parent])$
    end if
    if $grandParent \geq 1$ and $(heap[i] < heap[grandParent]$ or $heap[i] > heap[grandParent])$ **then**
      Swap$(heap[i], heap[grandParent])$

end if
$i \leftarrow parent$
end while
end procedure

Algorithm Sink($heap, i$)

---

procedure SINK($heap, i$)
$n \leftarrow \text{size}(heap)$
while lchild(i) $\leq n$ **do**
$left \leftarrow \text{lchild}(i)$
$right \leftarrow \text{rchild}(i)$
$target \leftarrow i$
if on min level and $heap[left] < heap[target]$ **then**
$target \leftarrow left$
else if on max level and $heap[left] > heap[target]$ **then**
$target \leftarrow left$
end if
if $right \leq \text{size}(heap)$ **then**
if on min level and $heap[right] < heap[target]$ **then**
$target \leftarrow right$
else if on max level and $heap[right] > heap[target]$ **then**
$target \leftarrow right$
end if
end if
if $target = i$ **then**
break
else
Swap($heap[i], heap[target]$)
$i \leftarrow target$
end if
end while
end procedure