2.

Priority queue

A data structure for maintaining a set S of elements, each with an associated value called a key

- A max-priority queue uses max-heap, and supports the following operations:
  - Insert (S, x, k): insert element x with key k into S
    Equiv. to: $S = S \cup \{x\}$
  - Maximum (S): returns element with largest key
  - Extract-Max(): removes & returns the element with largest key
  - Increase-key (S, x, k): Increases value of element x's key to the new value k ($k \geq x.key$)

Max-Heap-Maximum (A)

```
if A.heap-size < 1
    error "heap underflow"
return A[1]
```

Max-Heap-Extract-Max (A)

```
max = Max-Heap-Maximum(A)
A[1] = A[A.heap-size]
A.heap-size = A.heap-size - 1
Max-Heapify (A)
return max
```

$O(\lg n)$

Max-Heap-Increase-key (A, x, k)

```
if k < x.key
        error "New key is smaller than current key"
x.key = k
find the index i in array A where object x occurs
while i > 1 and A[Parent(i)].key < A[i].key
        exchange A[i] with A[Parent(i)]
        Update the information that maps Priority queue objects to array indices
        i = Parent(i)
```

$O(\lg n)$ + overhead of mapping Pr. que. obj to index

Max-Heap-Insert(A, x, n)

if A.heap-size == n

error "heap overflow"

A.heap-size = A.heap-size + 1

k = x.key

x.key = $-\infty$

A[A.heap-size] = x

map x to index heap-size in array

Max-Heap-Increase-key (A, x, k)

$O(\lg n)$ + overhead ...