設只允許下面三种 operations:

① 比較兩 floating point, 得到較大之值

② 比較兩 array entries 的 distance 和另兩 array entries 的 distance

③ swap 兩 array entries

並設皆為 unit cost: $O(1)$

①. <u>Nearest Neighbors:</u>

Input: $A[1,\dots,n]$ 為 unsorted array

Output: 兩 closest elements in $A[1,\dots,n]$

<u>idea:</u> 只要有 compare & swap operation 即可實現 comparison-based sorting

先將 $A[1,\dots,n]$ 排序為 $B[1,\dots,n]$

之後只比較相鄰元素之 distance

$\Rightarrow O(n \lg n)$

②. <u>Farthest Neighbor:</u>

Input: $A[1,\dots,n]$ 為 unsorted array

Output: 兩 farthest elements in $A[1,\dots,n]$

<u>idea:</u> 利用 compare 掃一次找 maxium

再掃一次找 minimum

$\therefore$ 為 $O(n)$

③. closest value to a floating point

Input: $A[1,\dots,n]$ 為 sorted array 和一 floating point x

Output: closest element to x in $A[1,\dots,n]$

<u>idea:</u> 利用 Binary search 找 x, 同時更新一值 value, 為記錄和 x distance 最小之值

同時儲當前的值和 x 之 distance

設 value 為 - global variable, 初值為 ∞

```
Binary_Search(A, x, i, j)
    m = $\frac{i+j}{2}$
    if A[m] == x
        value = min(value, abs(A[m]-x))
        return A[m]
    if A[m] > x
        value = min(value, abs(A[m]-x)
        Binary_Search(A, x, i, m-1)
    if A[m] < x
        value = min(value, abs(A[m]-x))
        Binary_Search(A, x, m+1, j)
```